EFFICIENT ALGORITHMS FOR SOLVING MULTI-OBJECTIVE OPTIMIZATION AND LARGE-SCALE TRANSPORTATION PROBLEMS

A Dissertation
Presented to
The Academic Faculty

By

Ian H. Herszterg

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology

August 2020

EFFICIENT ALGORITHMS FOR SOLVING MULTI-OBJECTIVE OPTIMIZATION AND LARGE-SCALE TRANSPORTATION PROBLEMS

Approved by:

Dr. Martin Savelsbergh, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Natashia Boland
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Alan Erera
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Mauricio Resende
Middle-Mile Planning, Research and Optimization Sciences Group
Amazon, U.S.

Dr. Alejandro Toriello
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: July 14, 2020

ACKNOWLEDGEMENTS

First, I would like to express my deep gratitude to my advisor, Prof. Martin Savelsbergh. During my PhD study, he has always been a source of knowledge and inspiration. I could not have asked for a greater advisor. I would also like to thank Prof. Alan Erera for his constant support and guidance throughout my years at Georgia Tech.

My sincere thanks to the members of my thesis committee, Prof. Alan Erera, Prof. Natashia Boland, Prof. Alejandro Toriello, and Dr. Mauricio Resende. I feel extremely honored to have the opportunity to present my thesis to you.

To all my friends in Atlanta, for all the craziness, laughs and all the fun we had in the last four years. I'll forever miss going out for late pizzas at Ray's and having drinks at Cypress St. It was a pleasure meeting all of you!

To my parents and my sister, for the unconditional love and support. Thank you for everything.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables viii

List of Figures xi

Chapter 1: Introduction and Background 1

1.1 Multi-Objective Optimization 3

1.2 Service Network Design for Same-Day Delivery 4

1.3 Incremental Network Optimization 5

1.4 Remark 7

Chapter 2: Fast and Robust Algorithms for Solving BOMIPs 8

2.1 Introduction 8

2.1.1 Contributions 11

2.2 Literature Review 13

2.3 Overview of existing methods 14

2.3.1 ϵ-Tabu Method 14

2.3.2 The Boxed Lined Method 14

2.4 Enhancements to existing methods 16

2.4.1 An enhanced implementation of the ϵ-Tabu Method 16
2.4.2 A purely lexicographic Boxed Line Method 20
2.4.3 A Hybrid Two-Phase Algorithm 23
2.4.4 Computational Study 25
2.4.5 Instances 25
2.4.6 Analysis 27
2.5 Approximating a mixed integer nondominated frontier 34
2.5.1 Computational Study 35
2.6 A parallel implementation of the Boxed Lined Method 38
2.6.1 Computational Study 40
2.7 Final remarks 42

Chapter 3: Heuristics for the Same-Day Delivery Problem with Hub Capacity Constraints 44

3.1 Introduction 44
3.1.1 Contributions 46
3.2 Literature Review 47
3.3 Problem Statement 48
3.4 Methodology 53
3.4.1 Metaheuristic Approach 53
3.4.2 Hybrid Matheuristic 63
3.5 Computational Study 65
3.5.1 Instances 65
3.5.2 Analysis 66

3.6 Final remarks 71

Chapter 4: Incremental Network Design with Multi-commodity flows 73

4.1 Introduction 73

4.1.1 Contributions 74

4.2 Literature Review 75

4.3 Problem Statement 77

4.4 Greedy Heuristics 81

4.4.1 By size of expansion 81

4.4.2 By largest reduction in flow costs 85

4.5 Exact method 91

4.5.1 Computing upper and lower bounds 93

4.6 Computational Study 98

4.6.1 Instances 98

4.6.2 Analysis 100

4.7 Incremental network design with temporary arc capacity expansions 108

4.8 The hub capacity expansion problem 110

4.8.1 Computational study 110

4.9 Final Remarks 116

Appendix A: Fast and Robust Algorithms for Solving BOMIPs 118

A.1 Pseudo-codes 118

A.2 Instance generation 121

A.3 Approximation results 123

Appendix B: Heuristics for the Same-Day Delivery Problem with Hub Capacity Constraints 127

References 137

LIST OF TABLES

2.1 Computational results for the Historical instances for variations of ϵTM. . . 29

2.2 Computational results for the Relaxed Historical instances for variations of ϵTM. 29

2.3 Computational results for the Rand and Bent instances for variations of ϵTM. 30

2.4 Computational results for variations of BLM. 31

2.5 Computational results for the Historical and Relaxed Historical instances for various solution methods. 33

2.6 Computational results for various solution methods for the Rand and Bent instances. 34

2.7 Avg Run times of 10 experiments for 21dat running PBLM-S. 41

2.8 Avg Run times of 10 experiments for 21dat running the different variants of PBLM. 43

3.1 Demand characteristics of the test instances 66

3.2 Network characteristics of the test instances 66

3.3 Performance of the Proposed Algorithms 68

3.4 Sensitivity Analysis for Operational Constraints 71

4.1 Instances generated for the incremental network design problem with multi-commodity flows 100

4.2 Computational results for the greedy heuristics. The best solutions are highlighted in bold. 102

4.3 Computational results when solving the integer programming formulation using CPLEX. 103

4.4 Computational results for DFSPE. 104

4.5 Upper and lower bounds for instance M1. 106

4.6 Computational results for the various algorithms 107

4.7 Original real-world instance and the reduced and adapted instances for the hub capacity expansion problem 113

4.8 Instances generated for the hub capacity expansion problem 114

4.9 Computational results for the greedy heuristics on the set of instances generated for the hub capacity expansion problem 114

4.10 Hub capacity expansions and flow costs at each period for E-INC-5 115

A.1 Approximation results for ϵTM. 124

A.2 Approximation results for PURELEX. 125

A.3 Approximation results for SPURELEX with $\rho = 0.005$. 126

A.4 Approximation results for the recursive variant of BLM. 126

B.1 MILS in-depth analysis of the 10 iterations of the outer loop for instance B with $\eta = 0$ 128

B.2 MILS in-depth analysis of the 10 iterations of the outer loop for instance B with $\eta = 0.005$ 128

B.3 MILS in-depth analysis of the 10 iterations of the outer loop for instance B with $\eta = 0.05$ 129

B.4 MILS in-depth analysis showing the average performance of the several neighborhoods for instance B with $\eta = 0$. 129

B.5 MILS in-depth analysis showing the average performance of the several neighborhoods for instance B with $\eta = 0.005$. 130

B.6 MILS in-depth analysis showing the average performance of the several neighborhoods for instance B with $\eta = 0.05$. 130

LIST OF FIGURES

2.1 The nondominated frontier of a BOMIP with minimization objectives. . . . 9

2.2 Correlation graphs between the area of the boxes solved in BLM and the solve time. . . . 21

2.3 Plot showing the area of the box and the solve time for the lexicographic IPs in ϵTM. . . . 24

2.4 Random cone width instances and the new bent instances 28

2.5 Approximation results for ϵTM, the recursive variant of BLM, PURELEX, and SPURELEX with $\rho = 0.005$. . . . 36

3.1 MILS: Binary Search Tree representing the loading operations over time at a loading dock of a hub 55

3.2 MILS: Initial and final states of the loading operations at a hub after the neighborhood move RE-ARRANGELOADINGPERIODS is applied. . . . 58

3.3 Average hub capacity utilization. . . . 70

4.1 Worst case example for GREEDYSIZEEXP-HL 83

4.2 Family of worst case instances for GREEDYSIZEEXP-HL 84

4.3 Worst case example for GREEDYSIZEEXP-LH 86

4.4 First iteration of GREEDYCOSTRED on the worst case example for the greedy heuristic by size of expansion 88

4.5 First iteration of GREEDYCOSTRED on the worst case example for the greedy heuristic by size of expansion 89

4.6 Bad example for GREEDYSIZECOSTRED 90

4.7 Example showing the benefit of allowing temporary arc capacity expansions in the network. 109

4.8 Transformation process from $G_H = (H, A_H)$ to $G = (N, A)$ for the hub capacity expansion problem 111

SUMMARY

The study of transportation problems has given rise to major developments in the fields of Applied Mathematics, Operations Research, and Industrial Engineering. However, the size of real-world instances often poses challenges for standard optimization algorithms and, thus, to the tractability of these problems. This calls for computationally efficient heuristics that can produce high quality solutions in a short period of time. Another layer of complexity in most real-world large-scale transportation problems is that planners have to take into account conflicting objectives when making decisions, most notably the minimization of transportation costs and the maximization of service quality. Often, the multiple objectives are converted into a single objective by linearly combining the objectives, which may not necessarily produce the desired solution. Approaches that produce Pareto optimal solutions are preferred, but are more involved and computationally intensive. With the set of Pareto solutions at hand, planners can make better decisions by analyzing the trade-offs in solution quality of assigning more or less priority to certain objectives.

Multi-objective integer linear programming problems have been studied for many decades in the field of Operations Research. There exist highly effective algorithms for generating the Pareto frontier (also known as the non-dominated frontier) of pure integer linear programs with two and three objectives. The situation is quite different for multi-objective *mixed* integer linear programs. Only a few computationally effective algorithms exist for bi-objective mixed integer programs (BOMIPs) – none for tri-objective mixed integer linear programs – and these have been proposed only relatively recently.

In this thesis, we address the two challenges highlighted above: solving multi-objective mixed integer programs and solving large-scale transportation problems. In Chapter 2, present a novel fast and robust algorithm for solving bi-objective mixed integer programs. The algorithm extends and merges ideas from two existing methods: the ϵ-Tabu Method and the Boxed Line Method. The result is a fast and, importantly, robust algorithm for

generating the non dominated frontier (NDF) of a BOMIP; it is robust in the sense that it works well across a wide range of instances with different characteristics. We demonstrate its efficacy in an extensive computational study. We also show that it is capable of producing a high-quality approximation of the NDF in a fraction of the time required to produce the complete NDF.

In Chapter 3, we study a new service network design problem encounter in a real-life urban same-day delivery system, in which the number of vehicles that can simultaneously load or unload at a hub is limited (hubs in densely populated metropolitan areas tend to be small due to high real-estate costs). Because of the presence of both time constraints for the commodities and capacity constraints at the hubs, it is no longer guaranteed that a feasible solution exists. First, we propose a non-trivial integer programming model for solving the problem. Second, to be able to solve real-world instances, we design and implement two heuristics: (1) a metaheuristic, and (2) a hybrid matheuristic, which takes advantage of the strengths of the metaheuristic and from an IP based heuristic approach. The efficacy of the heuristics is demonstrated on a number of real-world instances.

Finally, in Chapter 4, we study a novel incremental network design problem motivated by the expansion of hub capacities in package express service networks. We are given an existing and a target service network design, defined by a set of nodes, arcs, and origin-destination demands (commodities), and we seek to find a transition from the existing service network to the target service network, where the capacity of a subset of arcs has been increased. In each period, the capacity of a single arc can be increased and the cost in a period is given by the solution to an unsplittable multi-commodity flow problem. Our objective is to find a sequence of arc capacity expansions such that the total cost during the transition is minimized. We propose an integer programming model for solving the problem, as well as a set of greedy heuristics and an exact algorithm that explores the partial sequences of expansions in a depth-first search manner with a pre-specified order, where, for each partial sequence, we compute: (1) the cost associated with the partial sequence,

and (2) upper and lower bounds on the remaining costs of the transition sequence, used to curtail the enumeration of all possible sequences. We provide worst-case analyses for the greedy heuristics and we compare the efficacy of the algorithms to solving the integer programming formulation of the problem using a commercial solver. Finally, we use the proposed methodology to solve instances of the hub capacity expansion problem derived from real-world data from a large package express carrier. We also consider a variant of the problem in which temporary capacity expansions are allowed.

CHAPTER 1
INTRODUCTION AND BACKGROUND

The logistics and transportation industry is responsible for linking producers and consumers around the world through multiple modes of transportation, including air, maritime, and ground delivery services. Measured by revenue, package delivery has been one of the fastest growing segments of the freight transport business in the United States, with one representative player (UPS) reporting a revenue increase from $30 billion in 2000 to $72 billion in 2018. Two notable characteristics of the package delivery industry are that the dominant players, i.e., UPS and FedEx, operate very large networks and provide aggressive service offerings (e.g., next morning and next day delivery). Consequently, effective decision support for planning and operations is desired, but decision support models tend to be large and complex, and solving them in a reasonable amount of time tends to be challenging.

The study of transportation problems has given rise to major developments in the fields of Applied Mathematics, Operations Research, and Industrial Engineering. Today, many of these problems are concerned with finding the minimum cost of transporting a large set of commodities from a large number of sources (e.g.,distribution centers) to a large number of destinations (e.g.,warehouses, customers) under a set of operational and logistical constraints. The size of real-world instances often poses challenges for standard optimization algorithms and, thus, to the tractability of these problems. The computational challenges associated with large-scale instances make it virtually impossible to use commercial integer programming solvers. This calls for computationally efficient heuristics that can produce high quality solutions in a short period of time. Large-scale transportation problems discussed in the literature include ship scheduling and network design for cargo routing [1], railroad blocking problems [2], airline scheduling ([3],[4]), inventory routing problems

([5],[6]), service network design in freight transportation ([7],[8]), service network design for express package delivery services [9], and vehicle routing problems [10].

Another layer of complexity in most real-world large-scale transportation problems is that planners have to take into account conflicting objectives when making decisions, most notably the minimization of transportation costs and the maximization of service quality. Multi-objective transportation problems have been addressed in the literature (e.g., [11],[12],[13], and [14]). Often, the multiple objectives are converted into a single objective by linearly combining the objectives, which may not necessarily produce the desired solution. Approaches that produce Pareto optimal solutions are preferred, but are more involved and computationally intensive. Pareto optimal solutions have the property that there exists no other feasible solution that is at least as good in all objective values and better in at least one of them. With the set of Pareto solutions at hand, planners can make better decisions by analyzing the trade-offs in solution quality of assigning more or less priority to certain objectives. Approaches to find a (partial) set of Pareto optimal solutions have been proposed (e.g.,[15] and [16]).

Multi-objective integer linear programming problems have been studied for many decades in the field of Operations Research (see, for example, the surveys of [17] and [18], and the more recent paper of [19]). There exist highly effective algorithms for generating the Pareto frontier (also known as the non-dominated frontier) of pure integer linear programs with two and three objectives. The situation is quite different for multi-objective *mixed* integer linear programs (MOMIPs). Only a few computationally effective algorithms exist for bi-objective mixed integer programs (BOMIPs) – none for tri-objective mixed integer linear programs – and these have been proposed only relatively recently.

In this thesis, we address the two challenges highlighted above: solving multi-objective integer programs and solving large-scale transportation problems. We propose efficient and robust algorithms for solving bi-objective integer linear programming problems and efficient heuristics for solving large-scale problems arising in service network design for

package express carriers. Next, we provide some background information and list our contributions to the topics covered in the following chapters.

1.1 Multi-Objective Optimization

In multi-objective optimization, the goal is to generate a set of solutions that induces the nondominated frontier (NDF), also known as the Pareto front. The NDF is the set of non-dominated points (NDPs), where an NDP is a vector of objective values evaluated at a feasible solution with the property that there exists no other feasible solution that is at least as good in all objective values and is better in at least one of them.

For multi-objective *mixed* integer linear programs (MOMIPs), few algorithms exist that can effectively generate or approximate the non-dominated frontier. The most recent and most effective algorithms for solving BOMIPs are *criterion space search* methods, in which the search for the NDF operates in the space of the vectors of objective values, known as the criterion space. These methods take advantage of the advances in single-objective solver software, since they repeatedly solve single-objective problems, both linear programs (LPs) and mixed integer linear programs (IPs).

In Chapter 2, we first present an enhanced implementation of the ϵ-Tabu Method [20] that is $3.92\times$ faster, on average, than its original implementation on a set of benchmark instances often used in many published studies of BOMIP algorithms. Second, we present a novel fast and robust algorithm for solving bi-objective mixed integer programs. The algorithm extends and merges ideas from two existing methods: the ϵ-Tabu Method and the Boxed Line Method [21]. The result is a fast and, importantly, robust algorithm for generating the NDF of a BOMIP; it is robust in the sense that it works well across a wide range of instances with different characteristics. We demonstrate its efficacy in an extensive computational study. We also show that it is capable of producing a high-quality approximation of the NDF in a fraction of the time required to produce the complete NDF. Finally, we propose a parallel implementation of the Boxed Line Method that is on average $18.87\times$ faster

(using 20 threads) than its sequential implementation, as opposed to the modest average speed-up factor of 1.06 obtained when we allow the commercial solver to use all available threads for solving the optimization models in parallel.

1.2 Service Network Design for Same-Day Delivery

Driven by the growth of e-commerce, which relies heavily on faster delivery times [22], package express carriers have long sought to provide same-day delivery service in major metropolitan areas. Being able to do so is expected to increase demand (and profit) significantly given the compelling value proposition of same-day delivery for consumers [23].

To provide an economically viable (same-day) delivery service, carriers need to carefully allocate and utilize their resources. The challenge is to identify consolidation opportunities (so as to keep the costs down) while satisfying the service guarantees offered to customers (so as to maintain or increase market share). Service network design problems ([24],[25]) have long been used to aid in this process, and are usually modeled on a time-expanded network, especially when the time aspect is critical. The goal is to minimize cost through consolidation by choosing the right shipment paths in both space and time. A consolidation transportation system typically employs a complex hub network, in which vehicles transport packages between hubs, and packages are unloaded, sorted, and loaded at hubs. Routing packages through intermediate hubs is key to consolidation, but requires additional time and additional loading and unloading.

In Chapter 3, we study a new service network design problem encounter in a real-life urban same-day delivery system, in which the number of vehicles that can simultaneously load or unload at a hub is limited (hubs in densely populated metropolitan areas tend to be small due to high real-estate costs). Because of the presence of both time constraints for the commodities and capacity constraints at the hubs, it is no longer guaranteed that a feasible solution exists. First, we propose a non-trivial integer programming model for solving the problem. Second, to be able to solve real-world instances, we design and

implement two heuristics: (1) a metaheuristic, and (2) a hybrid matheuristic, which takes advantage of the strengths of the metaheuristic and from an integer programming based heuristic approach. The efficacy of the heuristics is demonstrated on a number of real-world instances. We observe that the metaheuristic approach is significantly faster than both the integer programming based heuristic and the hybrid matheuristic, but the efficiency comes at the price of serving fewer commodities. For the larger instances, the metaheuristic was able to find solutions under three hours serving only 3.7% less commodities than the hybrid matheuristic, whose running time exceeded 174 hours for the largest instance considered in our computational experiments. The hybrid matheuristic produces solutions in which the largest number of commodities are served, but also with the lowest costs when compared to the metaheuristic and the integer programming based heuristic.

1.3 Incremental Network Optimization

Less-than-truckload and package express transportation carriers regularly evaluate the design and operations of their service network. This can be prompted by a change in (forecast) demand or simply by a desire to identify cost savings (e.g., in the shuttle and line haul transportation costs or in the operational cost at hubs). However, changing the design and operations of an existing service network is not trivial and typically proceeds in two steps. First, planners decide where upgrades or expansions are introduced (e.g., where additional hubs will be located and where sorting capacity of existing hubs will be increased). Second, planners decide when to implement these upgrades and expansions. Given that a service network redesign may result in significant changes in the operations, the transition to a new target design typically occurs gradually, in a number of phases. Another reason for a gradual transition is that the change of the service network may require significant investments and budgets may force these investments to be made over a period of time. Given that there will be a gradual transition from an existing service network to a target service network, it becomes important to properly sequence the changes as this sequence

directly affects the operational costs and the profit during the entire transition period.

In incremental service network optimization, we assume that an initial service network design and a target service network design are provided, and we seek, in a fixed number of steps, to transition from the initial service network design to the target network design, where the goal is to maximize the total profits over the transition period (e.g., we want maximize the sum of the profits made in each of the transition periods).

In Chapter 4, we introduce a novel incremental network design problem motivated by the expansion of hub capacities in package express service networks: the *incremental network design problem with multi-commodity flows*. We model the problem as an integer program, propose and analyze greedy heuristics and develop an exact solution approach. In the greedy heuristics, we determine the sequence of expansions by selecting arcs based on: (1) the size of expansion (static), and (2) the largest immediate reduction in flow cost (dynamic). For finding optimal solutions, we propose a depth-first search partial enumeration algorithm that explores partial sequences of expansions in a depth-first search manner, where, for each partial sequence, we compute: (1) the cost associated with the partial sequence, and (2) upper and lower bounds on the costs of the remaining transition periods, which are used to curtail the enumeration of all possible sequences. We provide worst-case analyses for the greedy heuristics and we compare the efficacy of the algorithms to solving the integer programming formulation of the problem using a commercial solver. We show that the partial enumeration algorithm outperforms the commercial solver on a set of large instances when no feasible solutions are found by the commercial solver within a time limit of 24 hours. Finally, we use the proposed methodology to solve instances of the hub capacity expansion problem derived from real-world data from a large package express carrier. We also consider a variant of the problem in which temporary capacity expansions are allowed.

1.4 Remark

In addition to the research presented in this thesis, during my Ph.D. studies at Georgia Tech, I have also participated in a project on designing and implementing decision support technology to assist dispatchers in the daily management of loadplans in less-than-truckload service networks. The optimization technology developed in that project is presented in [26]. The focus is on how to effectively handle the fact that the freight volume that enters the service network on a particular day is highly likely to deviate from the forecast freight volume used to create the loadplan. These deviations cause delays when the capacity on planned freight paths is no longer sufficient, which, in turn, may result in missed service promises. Near real-time loadplan adjustments, i.e., rerouting freight on alternate paths, can improve on-time performance without incurring additional cost (e.g., without purchasing additional capacity). The problem of identifying effective alternate freight paths is modeled on a time-expanded network and fast heuristics are developed for its solution in order to ensure that there is sufficient time to put the adjusted loadplan in place. The loadplan adjustment technology is currently in use at a large US less-than-truckload carrier.

CHAPTER 2

FAST AND ROBUST ALGORITHMS FOR SOLVING BOMIPS

2.1 Introduction

In multi-objective optimization, the goal is to generate a set of solutions that induces the nondominated frontier (NDF), also known as the Pareto front. The NDF is the set of non-dominated points (NDPs), where an NDP is a vector of objective values evaluated at a feasible solution with the property that there exists no other feasible solution that is at least as good in all objective values and is better in at least one of them.

For multi-objective *mixed* integer linear programs (MOMIPs), few algorithms exist that can effectively generate or approximate the non-dominated frontier. The reason, in part, is that the NDF of a MOMIP can have a complex structure, e.g., it can contain open, half-open, and closed line segments, in addition to isolated points; see Figure 2.1 for an NDF of a BOMIP. The presence of these open, half-open, and closed line segments introduces many numerical challenges in the implementation of algorithms for generating the NDF of a BOMIP. The most recent and most effective algorithms for solving BOMIPs are *criterion space search* methods, in which the search for the NDF operates in the space of the vectors of objective values, known as the criterion space. These methods take advantage of the advances in single-objective solver software, since they repeatedly solve single-objective problems, both linear programs (LPs) and mixed integer linear programs (IPs).

We consider the bi-objective mixed integer linear program (BOMIP)

$$\min_{x \in \mathcal{X}} \{z(x) := (z_1(x), z_2(x))\} \tag{2.1}$$

where $z_1(x), z_2(x)$ are linear in x and the feasible region $\mathcal{X} \subseteq \mathbb{Z}^N \times \mathbb{R}^{n_C}$ is assumed to be nonempty and bounded. To differentiate between the integer and continuous components

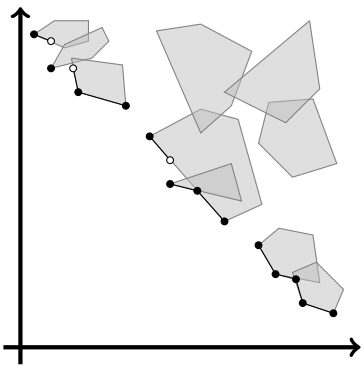


Figure 2.1: The nondominated frontier of a BOMIP with minimization objectives. The shaded regions represent the images of feasible solutions with a common integer part. The nondominated frontier is darkened.

of $x \in \mathcal{X}$, we use the convention $x = (x_I, x_C)$ where $x_I \in \mathbb{Z}^N$ and $x_C \in \mathbb{R}^{n_C}$. Let the projections of $\mathcal{X}$ onto the set of integer and real vectors be defined as $\mathcal{X}_I := \{x_I \in \mathbb{Z}^N : (x_I, x_C) \in \mathcal{X},\ \exists x_C \in \mathbb{R}^{n_C}\}$ and $\mathcal{X}_C := \{x_C \in \mathbb{R}^{n_C} : (x_I, x_C) \in \mathcal{X},\ \exists x_I \in \mathbb{Z}^N\}$, respectively. The feasible region $\mathcal{X}$ lies in the *decision space*, $\mathbb{R}^{N+n_C}$. The image of $\mathcal{X}$ under $z(\cdot)$, denoted by $\mathcal{Y} := \{y \in \mathbb{R}^2 : y = z(x),\ \exists x \in \mathcal{X}\}$, lies in the *criteria space*, $\mathbb{R}^2$.

For $x^1, x^2 \in \mathcal{X}$, if $z_i(x^1) \leq z_i(x^2)$ for $i = 1, 2$ and $z(x^1) \neq z(x^2)$, then $z(x^1)$ *dominates* $z(x^2)$. If $x^N \in \mathcal{X}$ and there does not exist $x \in \mathcal{X}$ such that $z(x)$ dominates $z(x^N)$, then $z(x^N)$ is a *nondominated point* (NDP) and x^N is *efficient*. The union of all nondominated points is the *nondominated frontier* (NDF), which we denote by $\mathcal{N}$.

A single NDP of (2.1) can be found by solving single-objective IPs[1] over $\mathcal{X}$ either by lexicographic optimization or by use of a scalarized objective function. The lexicographic IP hierarchically minimizes two objectives in turn. We denote the case of minimizing $z_1(x)$ and then $z_2(x)$ by

$$\eta = \text{lexmin}\{(z_1(x), z_2(x)) :\ x \in \mathcal{X}\}. \tag{2.2}$$

Solving (2.2) requires solving two IPs in sequence: $\eta_1 = \min\{z_1(x) :\ x \in \mathcal{X}\}$ and then $\eta_2 = \min\{z_2(x) :\ z_1(x) \leq \eta_1,\ x \in \mathcal{X}\}$ are solved, resulting in an NDP $\eta = (\eta_1, \eta_2)$ of

[1] We use the term integer program (IP) to refer to any single-objective problem that has integer variables, including mixed integer linear programs.

(2.1). In practice, the second IP tends to solve very quickly. For a given vector $\lambda \in \mathbb{R}^2_+$ we refer to

$$\min\{\lambda^T z(x) : \ x \in \mathcal{X}\} \tag{2.3}$$

as the *scalarized IP with respect to* λ. If x^λ is an optimal solution to (2.3) with $\lambda > 0$, so $\lambda_1, \lambda_2 > 0$, then $z(x^\lambda)$ is an NDP of (2.1). Not every NDP in the NDF can be found by such an IP: if, for a given NDP $z(x^N)$, there exists positive vector λ such that x^N is an optimal solution to (2.3), then the NDP is *supported*; otherwise, the NDP is *unsupported*.

The NDF can be described by nondominated line segments, vertical gaps, and horizontal gaps. Define $L(z^1, z^2)$ to be the line segment connecting endpoints $z^1, z^2 \in \mathbb{R}^2$, where the endpoints are ordered from left to right so that $z^1_1 \leq z^2_1$. The line segment may be open at both ends, half-open, or closed. Thus, we have that:

$$\{\xi z^1 + (1-\xi)z^2 : 0 < \xi < 1\} \subseteq L(z^1, z^2) \subseteq \{\xi z^1 + (1-\xi)z^2 : 0 \leq \xi \leq 1\}.$$

For each $i = 1, 2$, we refer to the endpoint z^i of $L(z^1, z^2)$ as *closed* if z^i belongs to the line segment, i.e., if $z^i \in L(z^1, z^2)$, and as *open* otherwise. In the case that $z^1 = z^2$, the line segment $L(z^1, z^2)$ consists of a single NDP. If all the points in $L(z^1, z^2)$ are nondominated and $L(z^1, z^2)$ is maximal, then we call $L(z^1, z^2)$ a *nondominated line segment (NLS)*.

The NDF may be described as a finite sequence of NLSs, $L(y^1, z^1), \ldots, L(y^K, z^K)$, say, for some $K \geq 1$, with $z^k_1 \leq y^{k+1}_1$ and $z^k_2 \geq y^{k+1}_2$ for all $k = 1, \ldots, K-1$, and for which:

$$y^1_1 < y^2_1 < \cdots < y^K_1 \qquad \text{and} \qquad y^1_2 > y^2_2 > \cdots > y^K_2.$$

A gap may appear between second and first endpoints, respectively, of two consecutive NLSs in the NDF: for some k, it may be that $z^k_1 < y^{k+1}_1$, which is a gap in the horizontal direction, and/or $z^k_2 > y^{k+1}_2$, which implies a gap in the vertical direction. In the case that $z^k_1 < y^{k+1}_1$, we define the interval $(z^k_1, y^{k+1}_1) \subset \mathbb{R}$ to be a *horizontal gap*. In this case, there exists no NDP p with $p_1 \in (z^k_1, y^{k+1}_1)$ and z^k must be an NDP and hence must be a closed

endpoint. In the case that $z_2^k > y_2^{k+1}$, we define the interval $(y_2^{k+1}, z_2^k) \subset \mathbb{R}$ to be a *vertical gap*. In this case, there exists no NDP p with $p_2 \in (y_2^{k+1}, z_2^k)$ and y^{k+1} must be an NDP and, hence, a closed endpoint. If there is a horizontal gap but no vertical gap between z^k and y^{k+1}, then y^{k+1} must be an open endpoint. If there is a vertical gap but no horizontal gap between z^k and y^{k+1}, then z^k must be an open endpoint.

Given $x_I \in \mathcal{X}_I$, the BOLP obtained from fixing the integer variables to x_I is called the *slice problem for* x_I [27]. The NDF of a slice problem consists of a (connected) set of (closed) line segments; we call this a *slice*[2]. The NDF of a slice problem for x_I is called the *slice for* x_I. The NDF of the BOMIP is contained in the union, over all $x_I \in \mathcal{X}_I$, of the slice for x_I. The NDF consists of all points in this union that are not dominated by any other point in the union.

2.1.1 Contributions

First, we propose an enhanced implementation of the ϵ-Tabu Method [20], in which we solve a single lexicographic IP based on a mixed integer programming model of piecewise linear functions. This single IP simultaneously considers multiple line segments of a slice to determine the "left-most" segment of the slice that is nondominated. That is, rather than investigating the line segments one by one, from "left to right", as in the original implemenation of ϵ-Tabu Method (described in more details in Section 2.4), we investigate them simultaneously, by solving a single lexicographic IP. The algorithm is $3.92\times$ faster, on average, than the original implementation ϵ-Tabu Method and $2.83\times$ faster, on average, than the Boxed Line Method on a set of benchmark instances often used in many published studies of BOMIP algorithms.

Second, we extend and merge ideas from two algorithms: the ϵ-Tabu Method and the Boxed Line Method [21]. The result is a fast and, importantly, robust algorithm for generating the NDF of a BOMIP; it is robust in the sense that it works well across a wide range

[2]Note that our definition of a slice differs from the original definition by [27], where it is defined as the *feasible set* for the slice problem as opposed to the resulting NDF.

of instances with different characteristics. We demonstrate that the algorithm can produce a high-quality approximation of the NDF in a fraction of the time it takes to generate the complete NDF. This is the first exploration of approximating the NDF of a BOMIP. We propose metrics to assess the quality of such an approximation.

Finally, we propose a parallel implementation of the Boxed Line Method that is $18.87\times$ faster than its sequential implementation, as opposed to the modest average speed-up factor of 1.06 obtained when we allow the commercial solver to solve the optimization models in parallel. The contributions of this research are summarized as follows:

- We propose an enhanced implementation of the ϵ-Tabu Method that significantly outperforms both its original implementation and the Boxed Line Method on a set of benchmark instances often used in many published studies of BOMIP algorithms;
- We extend and merge ideas from two algorithms: the ϵ-Tabu Method and the Boxed Line Method, which results in a fast and robust algorithm for generating the exact NDF of a BOMIP and for producing a high-quality approximation of the NDF in a fraction of the time it takes to generate the complete NDF, and;
- We propose a parallel implementation of the Boxed Line Method that achieves significant average speed-up factors when compared to its sequential implementation, as opposed to the modest average speed-up factor obtained when we allow the commercial solver to solve the optimization models in parallel.

The remainder of this chapter is organized as follows. In Section 2.2, we present a brief literature review on the most recent algorithms for solving multi-objective integer linear programs. In Section 2.3, we review two existing methods for generating the NDF of a BOMIP: the ϵ-Tabu Method and the Boxed Line Method. In Section 2.4, we propose enhancements to the ϵ-Tabu Method and the Boxed Line Method, which improve their performance and robustness, and present the results of an extensive computational study. In Section 2.5, we discuss how the proposed algorithms can be used to quickly produce an approximation of the NDF, then present metrics that assess the quality of that approxima-

tion. In Section 2.6, we present different parallel implementations for the Balanced Box Method, study the trade-offs between processing boxes concurrently and solving the optimization models in parallel, and finally present a computational study that investigates and compares the performances of the proposed implementations. We conclude, in Section 2.7, with some final remarks.

2.2 Literature Review

Multi-objective integer linear programming problems have been studied for many decades in the field of Operations Research (see, for example, the surveys of [17] and [18], and the more recent paper of [19]). This class of problems have been widely studied in the evolutionary algorithms community; see, for example, the surveys of [28], [29], and [30].

There exist highly effective criterion space algorithms for generating the exact Pareto frontier of pure integer linear programs with two and three objectives (e.g., [31] for bi-objective and [32] for tri-objective integer linear programs). The situation is quite different for multi-objective *mixed* integer linear programs (MOMIPs). Only a few computationally effective algorithms exist for bi-objective mixed integer programs (BOMIPs) – none for tri-objective mixed integer linear programs – and these have been proposed only relatively recently (e.g., [33], [20], [21], and [34]). Algorithms based on branch and bound focused in the space of the decision variables are given by [35] and [36]. Combining branching on the decision variables with Pareto branches, which work in the criterion space, [37] develop a method designed for the special case when only one of the two objective functions has continuous variables. In this case, the NDF only contains isolated points as no line segments can occur.

2.3 Overview of existing methods

2.3.1 ϵ-Tabu Method

The ϵ-Tabu Method (ϵTM) generates the frontier from left to right (or vice-versa). To initialize, ϵTM solves a lexicographic IP to find the upper left NDP of the frontier, $z^L = \text{lexmin}\{(z_1(x), z_2(x)) : x \in \mathcal{X}\}$. Next, ϵTM repeats two steps: (1) solve the slice problem for a given integer solution, and (2) check, sequentially – from left to right, if each line segment in the resulting slice is dominated or not using a (modified) lexicographic IP.

The latter step involves searching for the leftmost NDP "below" the line segment. If such an NDP is found, then ϵTM updates the rightmost endpoint of the line segment using the vertical projection of the new NDP onto the line segment; the line segment from the leftmost end of the line segment to the projected point is a NLS. ϵTM then processes the slice to which the new NDP belongs. If, on the other hand, no such NDP is found, then the line segment is a NLS. Hence ϵTM adds it to the frontier and proceeds to check the next line segment in the slice. If all line segments in the slice are confirmed to be nondominated, then ϵTM searches for the leftmost NDP "to the right" of the slice. If such an NDP is found, it defines the next slice. If no such NDP is found, the complete NDF has been found and ϵTM finishes. For more details see [20].

2.3.2 The Boxed Lined Method

The Boxed Line Method (BLM) [21] is an extension of the Balanced Box Method [31], which solves pure bi-objective integer programs to handle continuous variables. BLM has four main components: initialization, outer loop, inner loop, and line generation. To initialize, BLM solves two lexicographic IPs to find the upper left and lower right NDPs of the frontier. These define a rectangular region, or "box", that is added to a queue. The outer loop of BLM processes boxes in this queue until it is empty, at which time the algorithm terminates. The first step in processing a box is to search for the leftmost NDP in the lower

part of the box. This is achieved by solving a lexicographic IP constrained to the region of the criterion space below a horizontal line that splits the box.

If the NDP found lies strictly below the split line, the outer loop solves a mirrored lexicographic IP to find the rightmost NDP that is in the box and to the left of the NDP already found. Two new boxes are added to the queue with these NDPs as corner points. Otherwise the NDP lies on the split line and it must be that the split line intersects a NLS whose endpoints are yet unknown. To find these endpoints, BLM will first generate an *overestimate* (i.e., a superset) of the NLS by computing a line segment in a slice that contains the NDP. The inner loop is invoked to *refine* this line segment, eliminating dominated sections of it until it is proved to be a NLS. The original inner loop procedure in BLM accomplishes this by solving scalarized IPs; see [21] for the details.

In [21], two variants of BLM are presented. The *basic* variant, described above, has a simple inner loop that solves scalarized IPs. Additionally, a *recursive* variant is presented, which has a recursive inner loop: when a new NDP is found below the line segment, the inner loop recurses to discover the NLS containing the new NDP. The authors also present an enhancement designed for frontiers generated by very few slices (where each slice contributes many NLSs to the frontier). The *same-integer-solution* (SIS) enhancement subroutine is called only when the two corner points of a box are known to belong to the same slice (i.e., are generated by the same integer solution). The subroutine is called by the outer loop in place of splitting the box horizontally and solving scalarized IPs to obtain the nondominated portion of the line segment. Suppose the box $B(z^L, z^R)$ has both corner points belonging to the same slice. The enhancement checks if the line segment $L(z^L, z^R)$ dominates every other slice intersecting the box, in which case the entire frontier in the box belongs to that slice. To carry out this check, the algorithm solves a single scalarized IP (scalarized with respect to the gradient of $L(z^L, z^R)$) with one no-good constraint that excludes the integer solution associated with the two corner points. If the line segment dominates all other slices, the algorithm then switches over to dichotomic search to effi-

ciently generate the entire frontier by solving LPs. Otherwise, the enhancement returns a single NDP to the line generation procedure, and the inner loop then processes the resulting line segment as usual. This enhancement significantly reduces computation times for instances whose frontier contains very few slices.

Regardless of the variant used, BLM quickly partitions the criterion space into regions that are empty or dominated and boxes that are unexplored. If an iteration of the outer loop begins processing a box with area X, the sum of the areas of the new boxes added to the queue is at most $X/2$. Furthermore, since each box can be processed independently, BLM is easily parallelizable. These strengths facilitate a rapid approximation of the NDF.

2.4 Enhancements to existing methods

In this section, we propose enhancements to both ϵTM and the BLM variants, which improve their performance and robustness.

2.4.1 An enhanced implementation of the ϵ-Tabu Method

We propose an enhanced implementation of ϵTM, which we denote by ϵTM-PWL, in which we solve a single lexicographic IP based on a mixed integer programming model of piecewise linear ("PWL") functions. This single IP simultaneously considers multiple line segments of a slice to determine the "left-most" segment of the slice that is nondominated. That is, rather than investigating the line segments one by one, from "left to right", we investigate them simultaneously by solving a single lexicographic IP.

Suppose that from the first NDP found $z^0 = z(x^0)$ the slice for x_I^0 is generated and represented as a list of $K \geq 1$ line segments $\{L(z^0, z^1), L(z^1, z^2), \ldots, L(z^{K-1}, z^K)\}$ ordered from left to right. To check if a single line segment, say $L(z^i, z^{i+1})$, is (partially) dominated, where z^i is known to be nondominated, ϵTM solves the following lexicographic

IP:

$$\begin{aligned} \hat{z} = \text{lexmin} \quad & (z_1(x), z_2(x)) \\ \text{s.t.} \quad & z_1(x) \leq \lambda z_1^i + (1-\lambda) z_1^{i+1}, \\ & z_2(x) \leq \lambda z_2^i + (1-\lambda) z_2^{i+1}, \\ & x_I \neq x_I^0, \\ & x \in \mathcal{X}, \\ & 0 \leq \lambda \leq 1, \end{aligned} \tag{2.4}$$

where $x_I \neq x_I^0$ represents a no-good constraint. Observe that if (2.4) is feasible, then $\hat{z}$ is the leftmost NDP from a *different* slice that dominates some part of the line segment $L(z^i, z^{i+1})$.

For ϵTM-PWL, we propose a new formulation that resembles and is motivated by the modeling of a piecewise linear function using binary variables (see, for example, [38]). We introduce K binary variables y_i ($i = 1, \ldots, K$), one for each segment, with y_i associated with $L(z^{i-1}, z^i)$, and $K+1$ continuous variables λ_i ($i = 0, \ldots, K$), one for each of the end points of the line segments, with λ_{i-1} and λ_i associated with line segment $L(z^{i-1}, z^i)$.

We then express $z_1(x)$ and $z_2(x)$ as a convex combination of line segment end point coordinates (i.e., z_1^i and z_2^i for $i = 0, \ldots, K$), and set up a minimization problem that seeks the "left-most" point from a different slice that dominates any of the line segments. More specifically, we solve the following IP:

$$\hat{z}_1 = \quad \min z_1(x) \tag{2.5a}$$

$$\begin{aligned} \text{s.t.} \quad & z_1(x) = \sum_{i=0}^{K} \lambda_i z_1^i \\ & \sum_{i=0}^{K} \lambda_i = 1 \end{aligned} \tag{2.5b}$$

$$
\begin{aligned}
&\lambda_0 \leq y_1 \\
&\lambda_i \leq y_{i-1} + y_i \quad \forall i = 1, \ldots, K-1 \\
&\lambda_K \leq y_K \\
&\sum_{i=1}^{K} y_i = 1
\end{aligned}
\tag{2.5c}
$$

$$
z_2(x) \leq \sum_{i=0}^{K} \lambda_i z_2^i \tag{2.5d}
$$

$$
\begin{aligned}
&x_I \neq x_I^0 \\
&y_i \in \{0, 1\} \quad i = 1, \ldots, K \\
&0 \leq \lambda_i \leq 1 \quad i = 0, \ldots, K \\
&x \in \mathcal{X}
\end{aligned}
$$

where y_i for $i = 1, \ldots, K$ indicates whether the NDP found, if any, is "below" line segment $L(z^{i-1}, z^i)$ ($y_i = 1$) or not ($y_i = 0$).

If $z_1^{i-1} \leq z_1(x) \leq z_1^i$, then we want to express $z_1(x)$ as a convex combination of the z_1 coordinates of the end points of the line segment $L(z^{i-1}, z^i)$, i.e., $z_1(x) = \lambda_{i-1} z_1^{i-1} + \lambda_i z_1^i$ with $\lambda_{i-1} + \lambda_i = 1$. Constraints (2.5c) ensure that this happens as they force the selection of a single line segment, i.e., exactly one y_i will be equal to one, and force all but the two convex combination variables associated with the line segment, i.e., λ_{i-1} and λ_i, to be zero. Constraints (2.5d) ensure that the NDP identified either dominates or coincides with a point on the line segment. If the IP is infeasible, then the slice is nondominated (as there exists no feasible solution "below" its frontier). If the IP has a solution, then we continue the lexicographic minimization with

$$
\begin{aligned}
\hat{z}_2 = &\min z_2(x) \\
&z_1(x) \leq \hat{z}_1 \\
&x_I \neq x_I^0
\end{aligned}
$$

$$x \in \mathcal{X},$$

to ensure that we have a nondominated point, $\hat{z}$. By construction, there is a unique $j \in \{1, \ldots, K\}$ such that $z_1^{j-1} < \hat{z}_1 \leq z_1^j$. Then $L(z^{i-1}, z^i)$ is a NLS for all $i = 1, \ldots, j-1$ and $L(z^{j-1}, \tilde{z})$ is a NLS, where $\tilde{z}$ is the vertical projection of $\hat{z}$ onto $L(z^{j-1}, z^j)$. The proposed formulation adds $K + 3$ new constraints and $2K + 1$ new variables.

Note that one may decide to work with a partial slice, i.e., with the left-most $\hat{K} < K$ line segments obtained when solving the slice problem. That is, stop solving the slice problem after $\hat{K}$ line segments have been found and check if that portion of the frontier is dominated. If no dominated point is found, then resume solving the slice problem starting from the $(\hat{K}+1)^{th}$ point in the frontier, and repeat the process until we find either a dominating point or we reach the end of the frontier. Next, we propose two different schemes for dynamically adjusting the value of $\hat{K}$ throughout the execution of the algorithm in an attempt to reduce the computational effort required for solving the slice problems.

Let K_1 be the total number of segments of the first slice problem encountered in the algorithm and $\hat{K}_i$ be the number of line segments considered when solving the i^{th} slice problem in partial slices. Also, let $\bar{K}_i$ be the index of the line segment from the i^{th} slice problem, from left to right, such that the leftmost identified NDP was found below that line segment. We start by setting $\hat{K}_2 = K_1$ and adjust the values of $\hat{K}$ as follows:

- ϵTM-PWL(BS-N): If $\bar{K}_i < \hat{K}_i$ for the past N consecutive iterations (i.e., $\bar{K}_{i-N} < \hat{K}_{i-N}, \bar{K}_{i-N+1} < \hat{K}_{i-N+1}, \ldots, \bar{K}_i < \hat{K}_i$) then we set $\hat{K}_{i+1} = \hat{K}_i/2$. That is, if a NDP was found to the left of the $\hat{K}^{th}$ line segment in the previous N slice problems solved by the algorithm, then we reduce the number of line segments per partial slice in the next iteration of the algorithm by half. If, on the other hand, $\bar{K}_i > \hat{K}_i$ for the past N consecutive iterations, then $\hat{K}_{i+1} = \hat{K}_i \times 2$. In summary, we perform a binary type search on the optimal value for $\hat{K}$ by learning from the success and failure rates in previous iterations.

- ϵTM-PWL(AV-N): If either $\bar{K}_i < \hat{K}_i$ or $\bar{K}_i > \hat{K}_i$ for N consecutive iterations, we update $\hat{K}_{i+1}$ by taking the average between the past N consecutive values of $\bar{K}$ and $\hat{K}_i$, i.e., $\hat{K}_{i+1} = (\hat{K}_i + \frac{\sum_{j=i-N}^{i} \bar{K}_j}{N})/2$. This approach yields more conservative changes to the values of $\hat{K}$ when compared to ϵTM-PWL(BS-N).

2.4.2 A purely lexicographic Boxed Line Method

Recall that in the basic variant of BLM, the line generation procedure returns a line segment from a slice, which is then refined to the nondominated portion of the line segment by the inner loop that solves one or more scalarized IPs (see [21] for details).

Experiments with BLM have revealed that solving scalarized IPs can be computationally expensive; they are typically more expensive than solving a lexicographic IP (and sometimes *much* more expensive). Figure 2.2 shows, for three instances, Rand7500.A, Bent7500.A and C21 (to be described in more detail in Section 2.4.5), and for all lexicographic and scalarized IPs solved during the execution of the basic variant of BLM, the area of the active box when the IP is solved and the solve time of the IP. We observe that for all lexicographic IPs, the solve time is in the order of a few seconds, whereas for some of the scalarized IPs, the solve time is close to 500 seconds. This suggests that a variant of BLM that only solves lexicographic IPs (i.e., avoids solving scalarized IPs) may be faster on average or, at least, have more "stable" solve times.

Therefore, we explore the benefits of a Purely Lexicographic Boxed Line Method (PURELEX). PURELEX replaces the scalarized IPs solved when refining the initial overestimate of the line segment with lexicographic IPs. It does so in a way similar to ϵTM, using one lexicographic IP solve per endpoint. Given a line segment $L(z^1, z^2)$ containing a NDP $z^* = z(x^*)$ for some $x^* \in \mathcal{X}$ in its (relative) interior, let $\vec{w}$ represent the gradient of $L(z^1, z^2)$. PURELEX solves the following lexicographic IP to update z^1:

$$z^\alpha = \text{lexmin} \quad (z_2(x), z_1(x)) \tag{2.6}$$

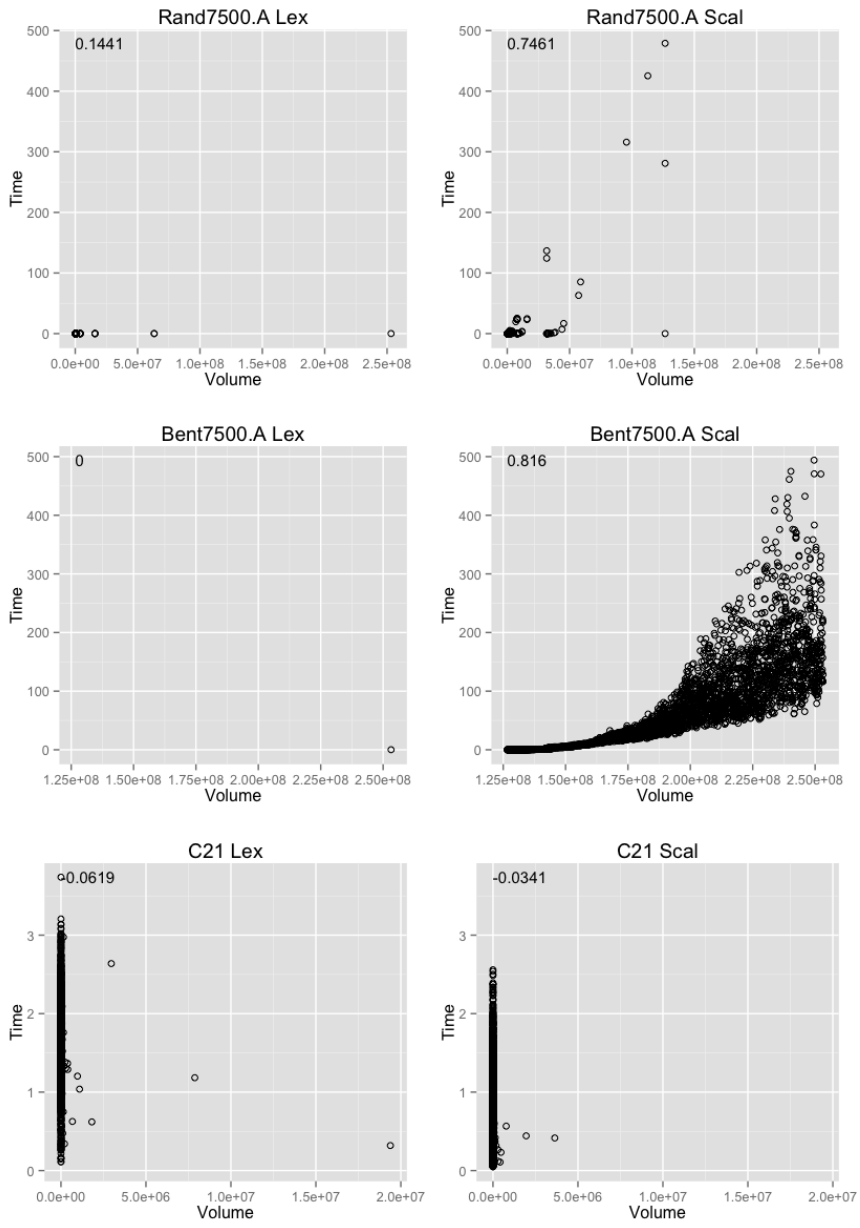


Figure 2.2: Correlation graphs between the area of the boxes solved in BLM and the solve time for lexicographic and scalarized IPs.For the basic variant of BLM, the area of the box and the solve time for the IPs, separated by instance and type of IP, are plotted as points. The resulting correlation coefficient is in the upper left of each graph. Note that only one lexicographic IP is solved on the bent instance, so the correlation is not well defined even though it is marked as zero. The correlation is at least 0.7 for scalarized IPs solved on the generated instances, i.e., Rand and Bent; everywhere else, the correlation is negligible.

$$
\begin{aligned}
\text{s.t.} \quad & \vec{w}^T z(x) < \vec{w}^T z^*, \\
& z_1(x) \leq z_1^*, \\
& z_2(x) \leq z_2^1, \\
& x_I \neq x_I^*, \\
& x \in \mathcal{X}.
\end{aligned}
$$

Note that (2.4) and (2.6) both model a lexicographic optimization of two objectives over a criterion space set with the same structure. In both cases, the criterion space set is defined by the intersection of three half spaces: two defined by the upper bounds on each objective and the third defined by the requirement to lie below a line (segment). They model this common structure in two different ways. Formulation (2.6) represents the three half-space constraints directly. Formulation (2.4), on the other hand, expresses them using a convex combination of the line segment endpoints, with a new continuous variable to model the weights in the convex combination. Both formulations impose a no-good constraint on the integer components of the solution ($x_I \neq x_I^*$ in the case of (2.6)). Unlike formulation (2.4), formulation (2.6) uses a strict inequality (implemented as $\vec{w}^T z(x) \leq \vec{w}^T z^* - \epsilon$, for some $\epsilon > 0$) to make the current line segment infeasible. In theory, therefore, the no-good constraint is redundant. However, computational experiments have shown that including the no-good constraint in (2.6) provides better numerical stability. When running ϵTM where (2.4) is replaced with (2.6), experiments indicate that instances are solved slightly faster with this new formulation (an average improvement of 1% in computational runtime). Note also that we solve either two single-objective IPs, and find an NDP that dominates a portion of $L(z^1, z^*)$, or one (infeasible) single-objective IP, and prove that $L(z^1, z^*)$ is nondominated.

If (2.6) is infeasible, then the endpoint z^1 does not need to be updated. Otherwise, the NDP z^α is used to update the endpoint z^1, using the horizontal projection of z^α onto $L(z^1, z^*)$. After the update, $L(z^1, z^*)$ is guaranteed to be nondominated (note that solving

the second IP in the lexicographic minimization is needed only because we require z^α to be nondominated in order to define the next box. To update the line segment, it suffices to solve only the first IP). A similar lexicographic IP is used to update z^2:

$$\begin{aligned} z^\beta = \text{lexmin} \quad & (z_1(x), z_2(x)) \\ \text{s.t.} \quad & \vec{w}^T z(x) < \vec{w}^T z^*, \\ & z_1(x) \leq z_1^2, \\ & z_2(x) \leq z_2^*, \\ & x_I \neq x_I^*, \\ & x \in \mathcal{X}. \end{aligned} \tag{2.7}$$

If (2.7) is infeasible, then endpoint z^2 does not need to be updated. Otherwise, the NDP z^β is used to update the endpoint z^2, using the vertical projection of z^β onto $L(z^*, z^2)$. After the update, $L(z^*, z^2)$ is guaranteed to be nondominated.

Thus, PURELEX identified the nondominated portion of a line segment by solving a minimum of two and a maximum of four single objective IPs, compared to one or more (possibly expensive) scalarized IPs in BLM. The pseudo-code for PURELEX can be found in Appendix A.1.

2.4.3 A Hybrid Two-Phase Algorithm

Reviewing the logic of ϵTM, we see that it is likely to solve fewer single-objective IPs than PURELEX, because when it shortens a line segment to its nondominated portion, it does so by solving either one (infeasible) IP or two single-objective IPs (one if the entire line segment belongs to the frontier and two when the line segment has to be shortened). However, solving fewer single-objective IPs will not always results in shorter solve times, because run time also depends on the time it takes to solve the single-objective IPs, which is impacted by the size of the box. The plots in Figure 2.3 show a clear correlation between

the solve time of a single-objective IP and the area of the active box. This explains, in part, why (as we shall see from the results in Section 2.4.6) ϵTM is more effective than PURELEX when the frontier lies in a "small" region of the criterion space, i.e, when the initial box $B(z^L, z^R)$ is "small". Another contributing factor is the fact that the frontier in a small region of the criterion space is likely to have fewer horizontal gaps, which is beneficial for ϵTM. Whenever ϵTM encounters a horizontal gap in the frontier, it needs to solve a lexicographic IP to find a new "starting" point, i.e., a new slice, which requires solving a lexicographic IP (and this lexicographic IP is not restricted to a small part of the box, which is usually the case when ϵTM determines whether a line segment belongs to the frontier or needs to be shortened).

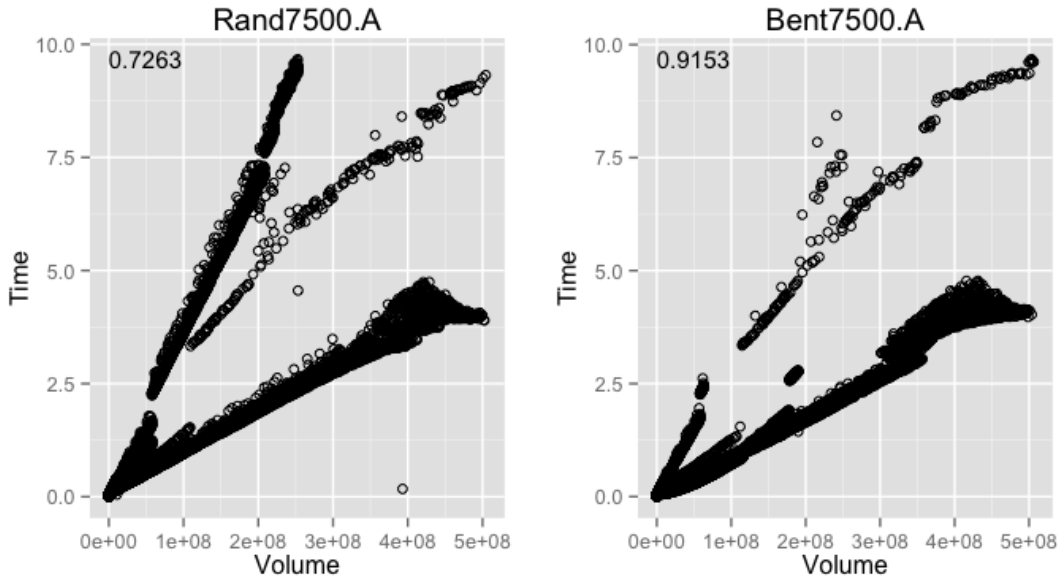


Figure 2.3: The area of the box and the solve time for the lexicographic IPs solved by ϵTM, separated by instance, are plotted as points. The resulting correlation coefficient is in the upper left of each graph.

This suggests that an algorithm that *switches* from PURELEX to ϵTM at some point during the generation of the frontier may be able to exploit the respective strengths of these methods and allay their respective weaknesses. We propose such an algorithm, which we call SPURELEX, as follows. SPURELEX starts by executing PURELEX to quickly decom-

pose the criterion space into many small as-yet unexplored boxes, and, then, when the total as-yet unexplored area of the criterion space becomes small, i.e., less than a fraction ρ ($0 < \rho < 1$) of the area of the initial box $B(z^l, z^U)$, it switches to ϵTM to rapidly generate the frontier in the remaining boxes (defining the as-yet unexplored area of the criterion space). We call SPURELEX-PWL the variant of SPURELEX using the ϵTM-PWL method. The pseudo-code for SPURELEX can be found in Appendix A.1.

2.4.4 Computational Study

The goal of our computational study is to demonstrate the efficacy of ϵTM-PWL and SPURELEX across a wide range of instances with different characteristics. All algorithms are coded in C++ and solve the linear and integer programs using IBM CPLEX Optimizer 12.6. All experiments were conducted in a single thread of a dedicated Intel Xeon ES-2630 2.3GHz with 50GB RAM, running Red Hat Enterprise Linux Server 7.4.

2.4.5 Instances

Three sets of instances were used: (1) the five largest instances (C320) from the benchmark instances proposed by [39], which we refer to as "Historical", (2) five modified versions of these instances, which we refer to as "Relaxed Historical", (3) three large randomly generated instances using the generation scheme proposed by [21], which we refer to as "Rand", and (4) three new randomly generated instances, which we refer to as "Bent".

The framework for the Historical instances, originally published by [39], includes m variables (half of them binary and half of them continuous), a randomly generated objective vector, and a set of m knapsack constraints with randomly generated weights and right hand sides. This framework was adapted to the bi-objective case in [33] by generating an additional objective vector, and since then this set of instances has been used in many published studies of BOMIP algorithms, including [20], [40], [34], and [21]. We include the largest instances in this set with $m = 320$. All instances in the set share similar features:

relatively few integer solutions contribute to the NDF, each of which mapping to a slice with many small line segments, and few interactions between distinct slices. This structure poses complications from an accuracy perspective, due to working with numerical rounding errors, as well as a homogeneous pool of benchmark problems.

By relaxing the constraints in the framework of the Historical instances, we aimed to induce more distinct integer solutions into the frontier as well as fewer miniscule NLSs. This led to the set of Relaxed Historical instances, which involved the following modifications. First, the constraints effecting the continuous variables only (approximately half of the constraint set) were removed. Second, the upper bound on the sum of n_b binary variables was increased from $n_b/3$ to $n_b/2$. Note that these modifications maintain the same number of integer and continuous variables (160 each) while reducing the number of constraints by approximately half.

[21] introduced an instance generation scheme to overcome the limitations of the Historical instances. The Rand instances were designed with the slices and final NDF in mind, then the BOMIP was "reverse-engineered." The slices in criterion space include: (1) a line segment traversing from the top left to the bottom right of the NDF and (2) boundaries of cones with vertices that dominate (a point in) the line segment (see Figure 2.4a). The NDF alternates between portions of the line segment and boundaries of each cone, which includes many intersections. Classes of these instances are defined by the number of vertices or cones chosen to dominate the line segment, which we simply call n. We include three relatively large instances of size $n = 7,500$ in this study, simply labeled "A", "B", and "C." Each of the n vertices is chosen randomly, so NDFs from instances of the same size still vary. For more details, see Appendix A.2.

We present a slight modification to the structure of the Rand instances that result in significant differences in computational comparison of BOMIP algorithms. The slice that is a line segment (1) is simply replaced with a "bent" line segment, i.e., two line segments, whose gradients have small difference, are joined at a central vertex (see Figure 2.4b). Note

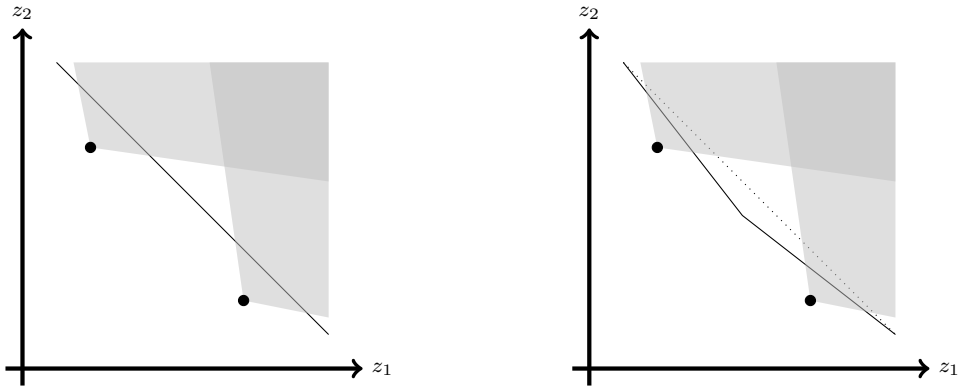
that the image of the feasible set restricted to this integer solution is now a cone, but one that is much wider than the cones associated with other integer solutions. Computationally, the slight difference in gradient vectors makes it much more difficult for algorithms that solve scalarized IPs, hence the motivation for replacing the IPs with lexicographic IPs in BLM. We include three instances with $n = 5000$, 7500, and 10000, respectively.

2.4.6 Analysis

In the tables with computational results, we report the following statistics: the number of nondominated points output by the algorithm (**nNDP**), the number of different integer solutions, i.e., the number of different slices appearing in the NDF (**nIPF**), the total time to discover the NDF (**TT**), the total time spent solving IPs (**IPT**), the total time spent solving LPs (**LPT**), the total number of IPs solved (**nIP**), the total number of lexicographic IPs (**nLex**), the total number of IPs solved during line restriction, i.e., as part of solving solving (2.6) and (2.7) in PURELEX and SPURELEX and as part of solving (2.7) in our implementation of ϵTM, (**RLIP**); the total number of scalarized IPs solved (**nScal**), the total number of IPs solved as part of the same-integer-solution enhancement, i.e., one such IP is solved per box with corner points generated by the same integer solution, (**nSIS**), the total number of LPs solved (**nLP**), the number of boxes processed (**nBox**), and, finally, the number of boxes with z^* on the horizontal split line, (**nZL**). When not applicable, an entry in the tables is marked with "-".

We start by evaluating the benefits of the enhancements to ϵTM. The results can be found in Tables 2.1-2.3. For these experiments, we only report the best variants of ϵTM-PWL(BS-N) and ϵTM-PWL(AV-N), i.e., ϵTM-PWL(BS-4) and ϵTM-PWL(AV-1).

Recall that the NDFs of the Historical and Relaxed Historical instances are composed of many small line segments, but are defined by a small number of integer solutions. On average, the number of integer solutions defining the frontier of the Historical instances is about 370, and each contributes about 50 line segments. In the Relaxed Historical in-



(a) The Rand instances include one slice in the form of a line segment (bold). In addition, for each of n additional integer solutions, its images of the feasible set is a cone (in gray) that dominates a portion of the line segment.

(b) The Bent instances include a slice that is a "bent" line segment (bold), which is actually the boundary of a very wide cone (i.e., two conjoined line segments).

Figure 2.4: Random cone width instances [21] and the new bent instances for $n = 2$.

stances, the average number of integer solutions defining the frontier is about 413, and each contributes about 33 line segments. The enhancements were specifically designed to exploit these situations, and explains why the enhanced versions of ϵTM perform so well on these instances. For the Historical instances, just by embedding the piecewise linear formulation into the lexicographic optimization models while checking if a slice is dominated, we see an average improvement of 68.55% in total running time when compared to the original implementation of ϵTM. This reduction in computational time is explained by the fact that this variant is solving, on average, 95.15% less IPs. The improvements are even greater when the slices are evaluated using ϵTM-PWL(AV-1), which balances the number of lexicographic minimization problems solved per slice and the difficulty of the lexicographic minimization problems that have to be solved. For these instances, we see that the solution time is reduced by an average factor of four. Similar improvements are also seen for the Relaxed Historical instances, where we see that the solution time is reduced by an average factor of 3.75 using the dynamic variants of ϵTM-PWL.

Unfortunately, these impressive gains cannot be observed for the Rand and Bent in-

Table 2.1: Computational results for the Historical instances for variations of ϵTM.

Instance	Algorithm	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	RLIP	nScal	nSIS	nLP	nBox	nZL
21		15636	295	11171.7	8574.0	2593.7	15923	43	15837	0	0	86952	1	0
22		18825	410	14106.2	11008.5	3092.8	19205	24	19157	0	0	98130	1	0
23	ϵTM	16420	343	12808.0	9758.0	3045.1	16752	25	16702	0	0	102407	1	0
24		18471	457	15331.1	12200.4	3123.4	18968	36	18896	0	0	101213	1	0
25		13216	337	9228.7	6843.2	2380.3	13518	21	13476	0	0	79127	1	0
21		15636	295	3370.1	871.1	2495.5	634	43	548	0	0	86952	1	0
22		18825	410	4114.2	1056.6	3053.7	846	24	798	0	0	98130	1	0
23	ϵTM-PWL	16420	343	4062.1	1051.5	3006.4	715	25	665	0	0	102407	1	0
24		18507	460	4829.1	1726.5	3098.6	1178	36	1106	0	0	101944	1	0
25		13216	337	3202.3	877.1	2322.6	697	21	655	0	0	79127	1	0
21		15636	295	2761.9	1354.8	1404.8	875	43	789	0	0	46539	1	0
22		18825	410	3443.1	1601.3	1839.0	1144	24	1096	0	0	57780	1	0
23	ϵTM-PWL(BS-4)	16420	343	3169.2	1595.4	1571.0	1002	25	952	0	0	51157	1	0
24		18484	460	4720.1	2414.2	2302.6	1377	36	1305	0	0	76611	1	0
25		13216	337	2600.8	1338.1	1260.4	959	21	917	0	0	40807	1	0
21		15636	295	**2727.6**	1341.4	1384.2	858	43	772	0	0	46152	1	0
22		18825	410	**3375.8**	1603.6	1769.3	1171	24	1123	0	0	56101	1	0
23	ϵTM-PWL(AV-1)	16420	343	**3165.7**	1620.5	1542.8	1005	25	955	0	0	50606	1	0
24		18490	460	**4112.0**	2437.1	1672.1	1528	57	1414	0	0	53777	1	0
25		13216	337	**2595.7**	1327.2	1266.1	923	21	881	0	0	40413	1	0

Table 2.2: Computational results for the Relaxed Historical instances for variations of ϵTM.

Instance	Algorithm	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	RLIP	nScal	nSIS	nLP	nBox	nZL
R21		14902	367	5437.3	3889.6	1544.7	15249	14	15221	0	0	93364	1	0
R22		15502	426	5196.9	3751.2	1442.6	15906	25	15856	0	0	86955	1	0
R23	ϵTM	15162	490	5677.5	3813.1	1861.3	15643	15	15613	0	0	110082	1	0
R24		15480	392	5590.9	4365.4	1222.5	15879	36	15807	0	0	70351	1	0
R25		11547	387	3777.3	2828.4	946.6	11919	30	11859	0	0	60953	1	0
R21		14902	367	2042.1	495.7	1544.0	753	14	725	0	0	93364	1	0
R22		15502	426	1934.0	493.5	1438.2	880	25	830	0	0	86955	1	0
R23	ϵTM-PWL	15168	491	2513.4	675.9	1834.6	1021	15	991	0	0	110333	1	0
R24		15501	394	1825.7	588.1	1235.5	883	36	811	0	0	70681	1	0
R25		11550	387	1405.3	466.0	937.7	826	30	766	0	0	60957	1	0
R21		14902	367	**1309.6**	580.3	727.8	1075	14	1047	0	0	43467	1	0
R22		15502	426	1282.4	516.8	764.1	1233	34	1165	0	0	46172	1	0
R23	ϵTM-PWL(BS-4)	15164	491	**1571.0**	783.8	785.6	1463	15	1433	0	0	44389	1	0
R24		15486	394	1572.5	774.0	796.9	1442	36	1370	0	0	45150	1	0
R25		11543	387	**1063.9**	529.6	533.0	1130	30	1070	0	0	35080	1	0
R21		14902	367	1339.7	559.2	779.0	1028	14	1000	0	0	46468	1	0
R22		15502	426	**1269.7**	491.1	777.0	1152	25	1102	0	0	46057	1	0
R23	ϵTM-PWL(AV-1)	15164	491	1631.2	781.4	848.2	1383	15	1353	0	0	47454	1	0
R24		15486	394	**1535.5**	690.3	843.5	1208	36	1136	0	0	47212	1	0
R25		11543	387	1081.0	530.6	549.2	1111	30	1051	0	0	35591	1	0

stances, as the NDF of these instances is not determined by a few integer solutions contributing many line segments, but by many integer solutions contributing with at most two line segments. In fact, we see that the overhead that comes with setting up and solving an integer program to determine the "left-most" segment of a slice that is nondominated is too large when slices consist of only a few line segments. For the largest Bent instance, the variant of ϵTM with the enhancement is unable to solve the instance within 24 hours.

Table 2.3: Computational results for the Rand and Bent instances for variations of ϵTM.

Instance		Algorithm	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	RLIP	nScal	nSIS	nLP	nBox	nZL
Rand	A	ϵTM	22502	7501	**3939.7**	3113.5	658.6	37149	369	36411	0	0	36403	1	0
	B		22508	7501	**4020.7**	3166.0	679.5	37135	382	36371	0	0	36365	1	0
	C		22506	7501	**3998.3**	3154.7	672.4	37135	381	36373	0	0	36370	1	0
	A	ϵTM-PWL	22502	7501	72226.8	71518.0	551.2	30008	369	29270	0	0	36403	1	0
	B		22508	7501	72617.2	71914.2	547.6	30009	381	29247	0	0	36363	1	0
	C		22506	7501	67619.6	66931.1	537.7	30011	381	29249	0	0	36370	1	0
	A	ϵTM-PWL(BS-4)	22502	7501	75828.4	74744.2	774.5	50032	369	49294	0	0	49981	1	0
	B		22507	7501	74231.6	73171.9	761.8	49927	382	49163	0	0	49883	1	0
	C		22505	7501	76525.6	75422.0	786.3	49954	381	49192	0	0	49900	1	0
	A	ϵTM-PWL(AV-1)	22502	7501	73539.3	72485.8	749.1	50032	369	49294	0	0	49981	1	0
	B		22508	7501	73689.3	72620.5	759.1	49927	382	49163	0	0	49883	1	0
	C		22505	7501	71821.5	70813.7	721.1	49954	381	49192	0	0	49900	1	0
Bent	5000.A	ϵTM	15003	5001	**1196.4**	834.6	295.9	24772	239	24294	0	0	24295	1	0
	7500.A		22504	7501	**2829.8**	1990.0	669.7	37145	370	36405	0	0	36403	1	0
	10000.A		30002	10001	**5207.2**	3719.0	1168.5	49525	490	48545	0	0	48546	1	0
	5000.A	ϵTM-PWL	15003	5001	27878.3	27530.0	278.5	20044	239	19566	0	0	24295	1	1
	7500.A		22504	7501	77702.0	76945.7	588.1	30041	370	29301	0	0	36403	1	0
	10000.A		-	-	-	-	-	-	-	-	-	-	-	-	-
	5000.A	ϵTM-PWL(BS-4)	15003	5001	27196.3	26723.5	350.1	33383	239	32905	0	0	33362	1	0
	7500.A		22504	7501	78361.7	77283.9	772.4	50011	370	49271	0	0	49967	1	1
	10000.A		-	-	-	-	-	-	-	-	-	-	-	-	-
	5000.A	ϵTM-PWL(AV-1)	15003	5001	29003.2	28501.7	372.9	33383	239	32905	0	0	33362	1	0
	7500.A		22504	7501	78495.6	77422.2	773.1	50011	370	49271	0	0	49967	1	0
	10000.A		-	-	-	-	-	-	-	-	-	-	-	-	-

Next, we evaluate the benefits of the variant of BLM in which only lexicographic IPs are solved. The results can be found in Table 2.4. We see that for the Rand and Bent instances, the PureLex variant substantially outperforms the basic version. The basic variant of BLM cannot solve any of the Bent instances in less than 24 hours, where PureLex can solve all of them in less than 3 hours. For the Historical instances, PureLex is still faster than the basic variant of BLM, but not by much. The opposite scenario is seen for the Relaxed Historical instances, where the basic variant of BLM is faster than PureLex, but again not by much.

As mentioned before, PURELEX solves two lexicographic IPs for every line segment, even when the line segment is nondominated, in which case a single scalarized IP would

Table 2.4: Computational results for variations of BLM.

Instance		Algorithm	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	RLIP	nScal	nSIS	nLP	nBox	nZL
Historical	21	BLM-Basic	15690	294	7932.3	6639.2	1291.5	7068	1890	40	1997	1251	44250	3127	1868
	22		18837	410	9885.0	8314.7	1568.6	9167	2470	23	2599	1605	53547	4041	2412
	23		16444	343	8948.5	7513.0	1434.0	7863	2120	26	2270	1327	47294	3434	2097
	24		18541	460	12168.6	10411.8	1755.0	10554	2832	34	3048	1808	56995	4626	2836
	25		13237	337	6649.4	5595.7	1052.4	7026	1896	27	1997	1210	38632	3074	1840
	21	BLM-Recursive	15725	294	8105.9	6690.0	1413.9	6547	1372	0	2736	1067	45180	2425	1351
	22		18856	410	9933.8	8173.8	1757.8	8221	1772	0	3292	1385	53660	3123	1716
	23		16463	343	9107.1	7504.5	1600.7	7094	1497	0	2958	1142	47519	2625	1474
	24		18562	460	12764.0	10848.8	1912.9	9630	2061	0	3934	1574	57447	3621	2066
	25		13248	337	6979.6	5768.6	1209.5	6417	1311	0	2786	1009	39408	2288	1256
	21	PURELEX	15685	295	**7160.8**	6026.8	1132.3	7918	1598	3493	0	1229	41643	2813	1577
	22		18824	410	**9237.9**	7843.9	1392.1	10127	2046	4464	0	1571	49956	3583	1990
	23		16439	343	**8268.2**	6966.2	1300.3	8656	1737	3859	0	1323	43876	3047	1714
	24		18526	460	**12517.5**	10928.4	1587	12802	2480	5870	0	1972	54581	4438	2529
	25		13230	337	**6008.5**	5072.1	934.9	7604	1533	3351	0	1187	35526	2688	1478
Relaxed Historical	R21	BLM-Basic	14904	366	3397.2	2694.2	701.9	7734	2088	12	2244	1302	43392	3361	2043
	R22		15522	425	3509.0	2751.5	756.3	9209	2472	31	2676	1558	46579	4009	2437
	R23		15164	491	4335.7	3486.1	848.3	10369	2818	18	3062	1653	48849	4457	2801
	R24		15525	392	3981.0	3169.1	810.7	8539	2296	34	2469	1444	46465	3728	2286
	R25		11552	387	2785.2	2258.6	525.7	7527	2041	30	2199	1216	35680	3235	2003
	R21	BLM-Recursive	14913	366	**2883.2**	2199.0	683.2	6670	1416	0	2754	1084	43014	2471	1368
	R22		15521	425	**3113.9**	2362.3	750.3	8250	1675	0	3585	1315	47103	2969	1642
	R23		15183	491	**3721.4**	2911.0	809.2	9172	1949	0	3842	1432	48849	3366	1931
	R24		15545	392	**3491.1**	2686.5	803.5	7734	1596	0	3327	1215	47182	2799	1588
	R25		11560	387	**2481.4**	1956.8	523.6	6814	1452	0	2862	1048	36037	2478	1414
	R21	PureLex	14896	366	3663.9	2987.2	675.5	8545	1724	3827	0	1270	40228	2965	1679
	R22		15503	425	3604.3	2897.7	705.3	9690	1931	4328	0	1500	42186	3410	1899
	R23		15164	491	4757.4	3961.8	794.0	11983	2383	5485	0	1732	45136	4101	2375
	R24		15513	393	4364.9	3574.6	789.0	9964	1952	4568	0	1492	43713	3433	1954
	R25		11547	387	3176.8	2666.8	508.8	8744	1741	3981	0	1281	33145	3004	1706
Rand	A	BLM-Basic	22502	7501	9106.7	5969.4	2942.9	73612	16970	680	34025	4967	162928	21884	21829
	B		22503	7501	12029.8	9157.3	2686.0	73704	17056	723	34036	4833	162990	21852	21815
	C		22503	7501	8215.6	5198.8	2825.5	73660	16983	718	34070	4906	163077	21851	21814
	A	BLM-Recursive	22504	7501	5747.7	3499.0	2052.2	44630	5485	0	33660	0	110886	5454	5425
	B		22504	7501	4843.9	2646.6	2008.9	44626	5331	0	33964	0	111047	5295	5261
	C		22505	7501	4849.6	2595.7	2057.9	44620	5302	0	34016	0	110910	5265	5230
	A	PURELEX	22502	7501	**4942.7**	2481.9	2246.8	116515	21896	72723	0	0	115994	21834	21766
	B		22502	7501	**4817.9**	2417.7	2191.3	116357	21850	72657	0	0	115849	21797	21742
	C		22499	7500	**4720.3**	2375.5	2141.8	116363	21847	72669	0	0	115868	21795	21745
Bent	5000.A	BLM-Basic	-	-	-	-	-	-	-	-	-	-	-	-	-
	7500.A		-	-	-	-	-	-	-	-	-	-	-	-	-
	10000.A		-	-	-	-	-	-	-	-	-	-	-	-	-
	5000.A	BLM-Recursive	15003	5001	84915.2	83826.7	986.1	29559	4996	0	19567	2	69979	4994	4994
	7500.A		-	-	-	-	-	-	-	-	-	-	-	-	-
	10000.A		-	-	-	-	-	-		-	-	-	-	-	-
	5000.A	PURELEX	15003	5001	**2135.7**	1063.0	982.3	77613	14622	48369	0	0	77506	14580	14534
	7500.A		22502	7501	**4739.5**	2367.2	2173.3	116393	21830	72733	0	0	115969	21798	21760
	10000.A		30002	10001	**8426.6**	4201.1	3864.7	155104	29214	96676	0	0	154927	29138	29056

have been sufficient to determine that the line segment is nondominated. However, because the solve times of the lexicographic IPs tend to be smaller and vary less, the overall increase in solve time is modest. For the Bent instances, where scalarized IPs can take prohibitively long, an algorithm free of scalarized IPs is fundamental to achieve good performance on the large instances.

Finally, we compare the performance of the two best performing variants of ϵTM and BLM with two variants of the hybrid, two-phase method: SPURELEX with $\rho = 0.005$ and SPURELEX-PWL with $\rho = 0.005$. The results can be found in Tables 2.5-2.6. The first thing to observe is that the enhancement to ϵTM still has a significant impact on the performance of the hybrid, two-phase method, for the Historical and Relaxed Historical instances, even though ϵTM is only used to find (part of) the NDF in a box that is relatively small (i.e., when the area is less than half a percent of the area of the original box). The second thing to observe is that PureLex does really well on the Historical instances and outperforms SPureLex-0.005, although not by much. Finally, as expected, we see that the version of SPureLex that does not use the enhanced version of ϵTM outperforms the one that does for the Rand and Bent instances, as the benefits of using the enhancement are insufficient to overcome the overhead incurred when using the enhancement. However, the difference is very small. These results indicate that SPureLex-PWL-0.005, the hybrid, two-phase method that uses PureLex in the first phase and used the enhanced version of ϵTM in the second phase, is the most robust and efficient method for solving BOMIPs.

It is interesting to observe that, compared to the variants of BLM, the number of times the Same-Integer-Solution enhancement is invoked by the variants of the hybrid, two-phase method is small. This indicates that a more careful tuning of the point at which the hybrid, two-phase methods switches methods may have some pay-off. By switching too early, the algorithm may miss out on opportunities to invoke the Same-Integer-Solution enhancement. Overall, ϵTM solves the smallest number of IPs for these instances. However, because solving lexicographic IPs for large-area boxes can be costly (as shown in Figure 2.3),

even though ϵTM solves significantly fewer IPs than PURELEX, ϵTM takes more time as it solves several IPs in boxes with large areas. This happens because ϵTM generates the nondominated frontier from left-to-right, and solves a lexicografic IP with respect to as-yet unprocessed portion of the line segment L (in case of the Rand instances) or $\hat{L}$ (in case of the Bent instances).

Table 2.5: Computational results for the Historical and Relaxed Historical instances for various solution methods.

Instance		Algorithm	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	RLIP	nScal	nSIS	nLP	nBox	nZL
Historical	21	ϵTM	15636	295	11171.7	8574.0	2593.7	15923	43	15837	0	0	86952	1	0
	22		18825	410	14106.2	11008.5	3092.8	19205	24	19157	0	0	98130	1	0
	23		16420	343	12808.0	9758.0	3045.1	16752	25	16702	0	0	102407	1	0
	24		18471	457	15331.1	12200.4	3123.4	18968	36	18896	0	0	101213	1	0
	25		13216	337	9228.7	6843.2	2380.3	13518	21	13476	0	0	79127	1	0
	21	ϵTM-PWL(AV-1)	15636	295	**2727.6**	1341.4	1384.2	858	43	772	0	0	46152	1	0
	22		18825	410	**3375.8**	1603.6	1769.3	1171	24	1123	0	0	56101	1	0
	23		16420	343	**3165.7**	1620.5	1542.8	1005	25	955	0	0	50606	1	0
	24		18490	460	**4112.0**	2437.1	1672.1	1528	57	1414	0	0	53777	1	0
	25		13216	337	**2595.7**	1327.2	1266.1	923	21	881	0	0	40413	1	0
	21	BLM-Recursive	15725	294	8105.9	6690.0	1413.9	6547	1372	0	2736	1067	45180	2425	1351
	22		18856	410	9933.8	8173.8	1757.8	8221	1772	0	3292	1385	53660	3123	1716
	23		16463	343	9107.1	7504.5	1600.7	7094	1497	0	2958	1142	47519	2625	1474
	24		18562	460	12764.0	10848.8	1912.9	9630	2061	0	3934	1574	57447	3621	2066
	25		13248	337	6979.6	5768.6	1209.5	6417	1311	0	2786	1009	39408	2288	1256
	21	PURELEX	15685	295	7160.8	6026.8	1132.3	7918	1598	3493	0	1229	41643	2813	1577
	22		18824	410	9237.9	7843.9	1392.1	10127	2046	4464	0	1571	49956	3583	1990
	23		16439	343	8268.2	6966.2	1300.3	8656	1737	3859	0	1323	43876	3047	1714
	24		18526	460	12517.5	10928.4	1587	12802	2480	5870	0	1972	54581	4438	2529
	25		13230	337	6008.5	5072.1	934.9	7604	1533	3351	0	1187	35526	2688	1478
	21	SPURELEX-0.005	15652	295	10541.5	9068.3	1469.0	16046	196	15642	0	12	44467	295	148
	22		18812	410	13674.0	11617.3	2051.6	19618	177	19262	0	2	61637	298	45
	23		16428	343	12204.8	10367.4	1832.9	17145	166	16811	0	2	50220	280	139
	24		18457	457	15075.8	12863.6	2207.4	19234	191	18845	0	7	62806	298	147
	25		13216	337	8645.6	7242.3	1399.9	13839	172	13487	0	8	43122	287	139
	21	SPURELEX-PWL-0.005	15652	295	3085.9	1683.9	1399.7	1522	196	1118	0	12	44467	295	148
	22		18812	410	3828.3	1879.6	1945.4	1731	177	1375	0	2	61637	298	145
	23		16428	343	3550.9	1816.0	1732.3	1550	166	1216	0	2	50220	280	139
	24		18482	460	4781.1	2702.1	2075.6	2056	191	1667	0	7	63157	298	147
	25		13216	337	2945.4	1584.2	1359.0	1554	172	1202	0	8	43122	287	139
Relaxed Historical	R21	ϵTM	14902	367	5437.3	3889.6	1544.7	15249	14	15221	0	0	93364	1	0
	R22		15502	426	5196.9	3751.2	1442.6	15906	25	15856	0	0	86955	1	0
	R23		15162	490	5677.5	3813.1	1861.3	15643	15	15613	0	0	110082	1	0
	R24		15480	392	5590.9	4365.4	1222.5	15879	36	15807	0	0	70351	1	0
	R25		11547	387	3777.3	2828.4	946.6	11919	30	11859	0	0	60953	1	0
	R21	ϵTM-PWL(BS-4)	14902	367	**1309.6**	580.3	727.8	1075	14	1047	0	0	43467	1	0
	R22		15502	426	**1282.4**	516.8	764.1	1233	34	1165	0	0	46172	1	0
	R23		15164	491	**1571.0**	783.8	785.6	1463	15	1433	0	0	44389	1	0
	R24		15486	394	**1572.5**	774.0	796.9	1442	36	1370	0	0	45150	1	0
	R25		11543	387	**1063.9**	529.6	533.0	1130	30	1070	0	0	35080	1	0
	R21	BLM-Recursive	14913	366	2883.2	2199.0	683.2	6670	1416	0	2754	1084	43014	2471	1368
	R22		15521	425	3113.9	2362.3	750.3	8250	1675	0	3585	1315	47103	2969	1642
	R23		15183	491	3721.4	2911.0	809.2	9172	1949	0	3842	1432	48849	3366	1931
	R24		15545	392	3491.1	2686.5	803.5	7734	1596	0	3327	1215	47182	2799	1588
	R25		11560	387	2481.4	1956.8	523.6	6814	1452	0	2862	1048	36037	2478	1414
	R21	PureLex	14896	366	3663.9	2987.2	675.5	8545	1724	3827	0	1270	40228	2965	1679
	R22		15503	425	3604.3	2897.7	705.3	9690	1931	4328	0	1500	42186	3410	1899
	R23		15164	491	4757.4	3961.8	794.0	11983	2383	5485	0	1732	45136	4101	2375
	R24		15513	393	4364.9	3574.6	789.0	9964	1952	4568	0	1492	43713	3433	1954
	R25		11547	387	3176.8	2666.8	508.8	8744	1741	3981	0	1281	33145	3004	1706
	R21	SPURELEX-0.005	14886	367	4917.8	4114.8	800.2	15397	178	15031	0	10	46496	314	156
	R22		15487	426	4772.0	3845.9	923.2	16315	193	15925	0	4	53921	334	168
	R23		15147	490	5128.6	4106.6	1019.1	16222	173	15875	0	1	54682	317	158
	R24		15466	392	5415.1	4522.2	890.0	16335	193	15945	0	4	49052	313	155
	R25		11553	387	3538.9	2960.4	576.2	12420	181	12055	0	3	36623	298	145
	R21	SPURELEX-PWL-0.005	14889	367	1527.7	718.8	807.6	1704	178	1338	0	10	46496	314	156
	R22		15487	426	1610.9	672.6	936.8	1886	193	1496	0	4	53921	334	168
	R23		15149	491	1829.9	832.5	995.7	1967	173	1620	0	1	54753	317	158
	R24		15487	394	1706.1	813.0	891.6	1817	193	1427	0	4	49306	313	155
	R25		11549	387	1166.6	608.0	557.5	1696	181	1331	0	3	36627	298	145

Table 2.6: Computational results for various solution methods for the Rand and Bent instances.

Instance		Algorithm	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	RLIP	nScal	nSIS	nLP	nBox	nZL
Rand	A	ϵTM	22502	7501	3939.7	3113.5	658.6	37149	369	36411	0	0	36403	1	0
	B		22508	7501	4020.7	3166.0	679.5	37135	382	36371	0	0	36365	1	0
	C		22506	7501	3998.3	3154.7	672.4	37135	381	36373	0	0	36370	1	0
	A	ϵTM-PWL	22502	7501	72226.8	71518.0	551.2	30008	369	29270	0	0	36403	1	0
	B		22508	7501	72617.2	71914.2	547.6	30009	381	29247	0	0	36363	1	0
	C		22506	7501	67619.6	66931.1	537.7	30011	381	29249	0	0	36370	1	0
	A	BLM-Recursive	22504	7501	5747.7	3499.0	2052.2	44630	5485	0	33660	0	110886	5454	5425
	B		22504	7501	4843.9	2646.6	2008.9	44626	5331	0	33964	0	111047	5295	5261
	C		22505	7501	4849.6	2595.7	2057.9	44620	5302	0	34016	0	110910	5265	5230
	A	PureLex	22502	7501	4942.7	2481.9	2246.8	116515	21896	72723	0	0	115994	21834	21766
	B		22502	7501	4817.9	2417.7	2191.3	116357	21850	72657	0	0	115849	21797	21742
	C		22499	7500	4720.3	2375.5	2141.8	116363	21847	72669	0	0	115868	21795	21745
	A	SpureLex-0.005	22502	7501	**1790.9**	865.2	821.7	38357	583	37191	0	434	38084	40	213
	B		22502	7501	**1782.6**	863.7	816.3	38353	594	37165	0	434	38042	42	215
	C		22502	7501	**1786.6**	866	813.1	38354	597	37160	0	434	38032	46	211
	A	SPureLex-PWL-0.005	22502	7501	1832.3	904.3	810	31343	583	30177	0	434	38084	40	213
	B		22502	7501	1820.3	901	804.3	31351	594	30163	0	434	38042	42	215
	C		22502	7501	1837.8	909.7	810.4	31346	597	30152	0	434	38032	46	211
Bent	5000.A	ϵTM	15003	5001	1196.4	834.6	295.9	24772	239	24294	0	0	24295	1	0
	7500.A		22504	7501	2829.8	1990.0	669.7	37145	370	36405	0	0	36403	1	0
	10000.A		30002	10001	5207.2	3719.0	1168.5	49525	490	48545	0	0	48546	1	0
	5000.A	ϵTM-PWL	15003	5001	27878.3	27530.0	278.5	20044	239	19566	0	0	24295	1	1
	7500.A		22504	7501	77702.0	76945.7	588.1	30041	370	29301	0	0	36403	1	0
	10000.A		-	-	-	-	-	-	-	-	-	-	-	-	-
	5000.A	BLM-Recursive	15003	5001	84915.2	83826.7	986.1	29559	4996	0	19567	2	69979	4994	4994
	7500.A		-	-	-	-	-	-	-	-	-	-	-	-	-
	10000.A		-	-	-	-	-	-		-	-	-	-	-	-
	5000.A	PureLex	15003	5001	2135.7	1063.0	982.3	77613	14622	48369	0	0	77506	14580	14534
	7500.A		22502	7501	4739.5	2367.2	2173.3	116393	21830	72733	0	0	115969	21798	21760
	10000.A		30002	10001	8426.6	4201.1	3864.7	155104	29214	96676	0	0	154927	29138	29056
	5000.A	SpureLex-0.005	15003	5001	**836.2**	402.1	386.8	25988	449	25090	0	39	25969	428	212
	7500.A		22502	7501	**1824.5**	882.5	838	38350	586	37178	0	45	38041	434	208
	10000.A		30002	10001	**3100.2**	1504.2	1416.8	50711	704	49303	0	52	50286	436	216
	5000.A	SPureLex-PWL-0.005	15003	5001	841.9	413.4	376.3	21300	449	20402	0	39	25969	428	212
	7500.A		22502	7501	1876.8	923.9	835.5	31322	586	30150	0	45	38041	434	208
	10000.A		30002	10001	3206.9	1583.1	1421.9	41319	704	39911	0	52	50286	436	216

2.5 Approximating a mixed integer nondominated frontier

In the literature on multi-objective optimization, an *approximation* of the NDF can take any of several different forms. Here, we use it to describe a subset of the NDF. In particular, we compare the parts of the NDF discovered by alternative exact algorithms prior to their completion. To do so, we require metrics that measure the quality of such an approximation. We introduce the following metrics specifically designed to assess the quality of a subset of the NDF as an approximation to the full NDF of a bi-objective mixed integer program:

1. The as-yet unexplored area of the criterion space (i.e., the area of the criterion space that may still contain parts of the frontier) as a fraction of the area of the initial box $B(z^L, z^R)$;
2. The fraction of isolated NDPs found;
3. The fraction of NLSs found;
4. The fraction of the total length of the NLSs found; and

5. The fraction of slices that contribute to the frontier found.

Note that metrics 2, 3, 4, and 5 are relative to the complete nondominated frontier, which is assumed to be known.

When computing the complete NDF, the order in which boxes are processed is immaterial. However, processing the boxes in the queue in nonincreasing order of their area has two advantages when computing an approximation of the NDF: (1) it naturally introduces diversification, in the sense that different parts of the criterion will be explored, and (2) the unexplored area of the criterion space reduces as fast as possible. Therefore, terminating the algorithm after a fixed amount of time, or terminating the algorithm when the unexplored area of the criterion space drops below a certain threshold (e.g., below some fraction of the area of the initial box $B(z^L, z^U)$), are both effective strategies to quickly produce a high-quality approximation of the NDF.

2.5.1 Computational Study

The goal of our computational study is to investigate SPURELEX's ability to efficiently produce high-quality approximations to the NDF. We compare the performance, in terms of their ability to produce high-quality approximations of the NDF, of ϵTM, the recursive variant of BLM, PURELEX, and SPURELEX with $\rho = 0.005$. Rather than enforcing early termination of an algorithm to obtain an approximation of the NDF, we report statistics of the approximation of the NDF at different points in time during the execution of the algorithms. We do this for three of the instances presented in Section 2.4.5; one from each of the set of historical, bent, and rand instances: 21, Bent7500.A, and Rand7500.A, respectively. The results are representative of what happens for the other instances in the corresponding set. The results can be found in Tables A.1 – A.4 in Appendix A.3, and are summarized in Figure 2.5.

We observe that, as expected, PURELEX and SPURELEX produce a high-quality approximation of the NDF much more quickly than ϵTM. In less than 10 minutes, PURELEX

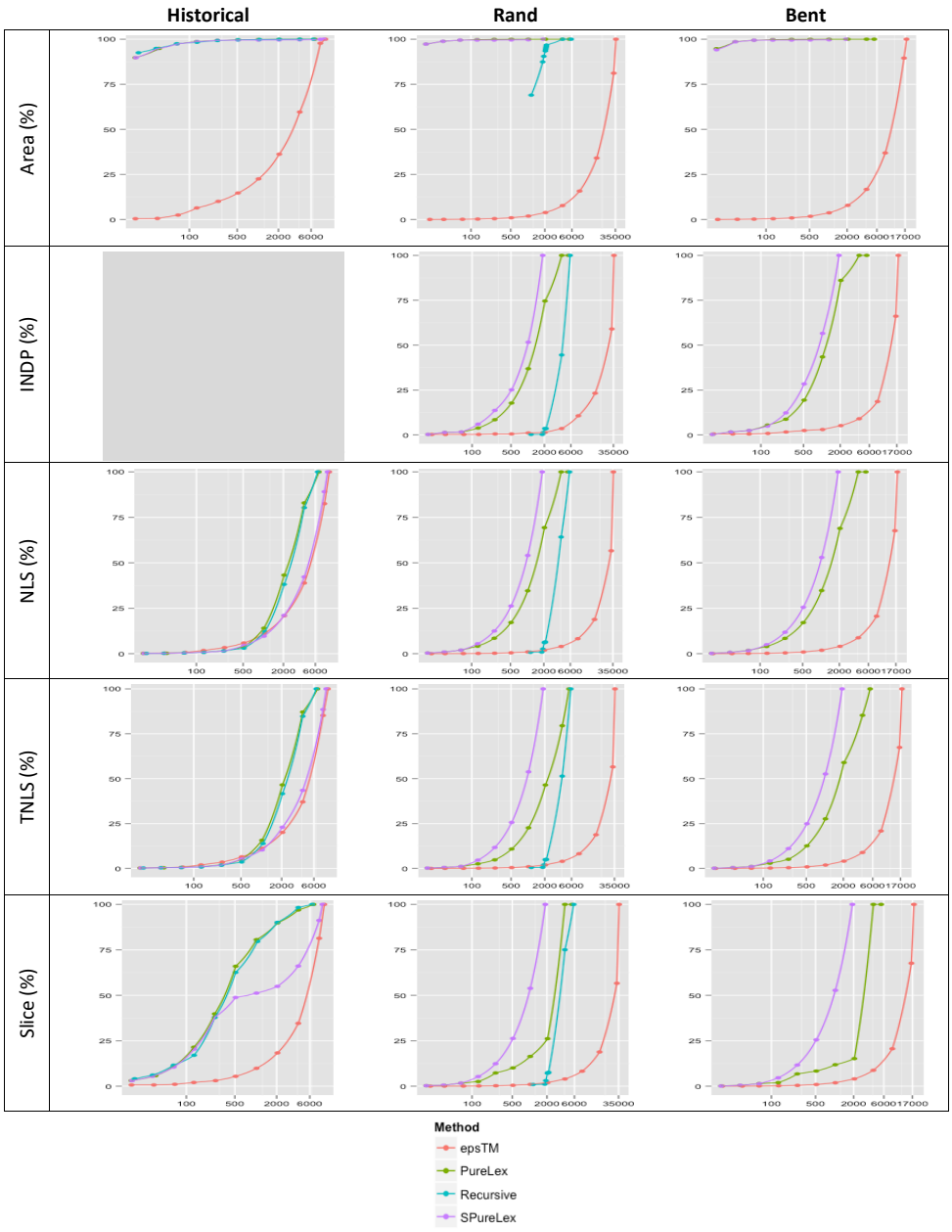


Figure 2.5: Approximation results for ϵTM, the recursive variant of BLM, PURELEX, and SPURELEX with $\rho = 0.005$. There are no isolated NDPs in the NDF of the historical instance, so this space is blank. (No approximation metrics are reported for the recursive variant of BLM on the bent instance.)

and SPURELEX have explored more than 99.5% of the area of $B(z^L, z^U)$, whereas ϵTM has explored a little more than 5% for the historical instance, and 1% or less for rand and bent instances.

Interestingly, looking at the fraction of the resolved area of the initial box $B(z^L, z^R)$ (**Area**), the fraction of the number of nondominated line segments found (**NLS**), the fraction of the total length of the nondominated line segments found (**TNLS**), and the fraction of slices that contribute to the frontier found (**Slice**) for rand and bent instances, we observe that ϵTM advances very slowly at the beginning (from an approximation perspective) and speeds up towards the end. This is due, in part, to the fact that in the beginning, the boxes cover a large area in criterion space, and the lexicographic IPs are time-consuming. As the area of the boxes decreases, the lexicographic IPs are solved faster, and, consequently, the statistics improve more rapidly. For the historical instance, this is less noticeable because the solve times for the lexicographic IPs tend to be smaller (as the generated line segments are small).

Even though the recursive variant of BLM can be competitive when it comes to producing the complete nondominated frontier, it is not the ideal candidate for producing approximations. For the historical instance, there is never any recursion, so the algorithm produces high-quality approximations throughout the execution. However, on the rand instance, the first reported data point occurs late during the execution because the algorithm runs deep into recursion and only reports approximation metrics once it has returned to depth level zero. This is also the reason why no approximation metrics are reported for the bent instance (the entire execution time is spent on the first recursion without returning to depth level zero). After the algorithm returns to depth level zero for the first time, it quickly approximates the rest of the frontier, but the depth of recursion and resulting lag time of reporting makes the recursive variant of BLM less effective for approximating nondominated frontiers. Another interesting observation is that PURELEX and SPURELEX quickly find a large fraction of the slices that contribute to the frontier for the historical instance. This is a

consequence of the structure of the nondominated frontier, in which each slice contributes many nondominated line segments to the frontier, and, because PURELEX and SPURELEX quickly decompose the criterion space into small boxes, they tend to find NDPs from different slices. Also observe from the historical instance, however, that once SPURELEX switches to ϵTM, the algorithm discovers new slices more slowly, at a rate comparable to ϵTM on the same instance. This is the trade-off: sometimes the switch from PURELEX to ϵTM improves performance (e.g., rand and bent instances), and sometimes it worsens performance (e.g., historical instances), but overall the switching makes for a more *robust* algorithm. These results show that also when it comes to finding approximate nondominated frontiers, SPURELEX (or one of its variants) is fast and robust and the algorithm of choice.

2.6 A parallel implementation of the Boxed Lined Method

As mentioned in Section 2.3.2, the boxes generated by BLM can be processed independently. However, it is important to have an efficient scheme for managing the workload and the resources shared between threads (the multiple lines of execution running concurrently and sharing resources). Given that each box can produce up to two new boxes, threads will, in most part, work asynchronously and spend a considerable amount of time in an idle status waiting for new jobs.

Increasing the number of threads for solving optimization problems in commercial solvers is most likely to reduce execution times. However, the existence of sequential algorithmic steps and the of parallelism efficiency factor limits the execution time improvement. As stated in the documentation of the commercial solver used in our experiments, when the thread count increases, the parallel algorithm becomes less and less efficient. This is due to synchronization steps between threads and thread contention over computational tasks. Assigning a large number of threads for solving small optimization models can, in some cases, increase the computational effort required for solving the problem since many threads will

quickly become idle and these extra synchronization steps for managing threads will not pay off. Moreover, in a multithread execution, each thread requires a copy of the model data, leading to higher memory usage by the algorithm.

Motivated by these observations, we propose different methodologies for parallelizing the basic variant of BLM and study the trade-offs between assigning different numbers of threads for processing boxes concurrently and for the commercial solver used for solving the optimization models in parallel. We propose three different parallel implementations of the basic variant of BLM (PBLM), where threads will share and collect jobs using:

- a single *shared* queue (PBLM-S) between all threads: each thread will collect and place boxes in a single shared queue. Whenever a thread finishes processing a box, it waits until either: (i) a new box is placed in the share queue, or (ii) all threads finish processing a box and no more boxes are placed in the shared queue. In this case, the algorithm terminates.
- a single *local* queue for each thread and a single *shared* queue between all threads (PBLM-L): each thread starts by collecting a box from the shared queue and will only place new boxes in its own local queue if the size of the queue is less than or equal to a fixed threshold N_B. Whenever there are N_B boxes in its local queue, the thread will then place new boxes in the shared queue. When there are no more boxes to process in its local queue, the thread will wait until either: (i) a new box is placed in the shared queue, or (ii) all threads finish processing boxes in their local queues and no more boxes are placed in the shared queue. In this case, the algorithm terminates. The choices of N_B are discussed in more detail in Section 2.6.1.
- *shared local* queues between threads (PBLM-SL), one for each thread: each thread starts by collecting any available box from the collection of shared local queues and will always place new boxes in its own shared local queue. Whenever a thread finishes processing all boxes in its local queue, it will search for the closest available box from the one it just processed (i.e., by computing the euclidean distance between

the bottom-left corner of each box in the shared local queues and the bottom-left corner of the box it just processed). If no boxes are found, the thread will wait until either: (i) a new box is placed in any of the shared local queues, or (ii) all threads finish processing boxes in their local queues and no more boxes are generated. In this case, the algorithm terminates.

While PBLM-S is a more traditional and baseline approach, PBLM-L and PBLM-SL enforce that threads work in a same region of the criterion space as much as possible, such that solvers can use the information acquired from models previously solved to give a warm start for the models in the next iteration. Having an incumbent solution from the very beginning of the optimization routine allows solvers to eliminate portions of the search space, potentially leading to reduced computing times for finding solutions. By sequentially processing boxes covering overlapping regions of the criterion space, we can take advantage of the information obtained by solvers in past iterations and accelerate the optimization of new models.

2.6.1 Computational Study

The goal of our computational study is twofold. First, we want to analyse the trade-offs and the different speed-up factors obtained when assigning different numbers of threads for processing boxes concurrently and for the commercial solver. Second, we want to investigate and compare the performance of the three variants of PBLM. The algorithms are coded in C++ using the POSIX pthread standard API and solve the linear and integer programs using IBM CPLEX Optimizer 12.6. All experiments were conducted in the same dedicated Intel Xeon ES-2630 2.3GHz with 20 physical CPU cores, 50GB RAM, running Red Hat Enterprise Linux Server 7.4.

Table 2.7 shows results for the average run time over 10 experiments for instance 21 of the set of Historical instances presented in Section 2.4.5 considering different settings for the number of threads assigned for processing boxes concurrently (using PBLM-S)

and for the commercial solver for solving the optimization models in parallel. We report: the number of threads assigned to BLM (N_{BLM}), the number of threads assigned to the commercial solver (N_S), the average running time of the 10 experiments (**Avg. TT**), and the average speed-up factor of each setting with respect to the sequential implementation of BLM (**Avg. Speed-Up**). The results are representative of what happens for the other instances in the corresponding set.

The results show no real benefit in assigning a larger number threads to the commercial solver, and in fact, by doubling the number of threads from 10 to 20, we see a 5% increase in average running time. Assigning more threads to BLM, on the other hand, shows a significant reduction in computational effort. We reach an average speed-up factor of 17.17 by assigning all available threads to process boxes concurrently, as opposed to the modest average speed-up factor of 1.06 obtained when assigning more threads for solving the optimization models in parallel. These are not surprising results, given that for this instance BLM solves 7068 IPs which are relatively small in size (320 constraints and 160 variables) and little is gained from using multi-threading to solve these models. Solving multiple small IPs concurrently yields much larger improvements in total computational time, showing the benefit of parallelizing BLM over the commercial solver.

Table 2.7: Avg Run times of 10 experiments for 21dat running PBLM-S.

Instance	N_{BLM}	N_S	**Avg. TT**	**Avg. Speed-Up**
21dat	1	1	7932.29	-
	1	2	7602.50	1.04
	1	4	7584.31	1.04
	1	8	7431.41	1.06
	1	10	7411.12	1.06
	1	20	7798.72	1.01
	5	4	1300.27	6.10
	10	2	691.42	11.47
	20	1	461.99	17.17

Next, we compare the average execution times of 10 experiments for the three variants of PBLM using different settings for the number of threads assigned to BLM and to the

commercial solver. We report: the variant of PBLM used (**Algorithm**), the average speed-up factor of each algorithm with respect to the baseline approach, PBLM-S (**Avg. Speed-Up**) and the average percentage of work time per thread, i.e., the average percentage of time spent by each thread solving boxes (**Avg. Work(%)**). The results shown in Table 2.8 are again representative of what happens for the other instances in the corresponding set.

Although the speed-up factors reported with respect to the baseline approach are not that impressive (up to a factor of 1.11), we see that PBLM-LS has a clear advantage over the other two approaches, especially when compared to PBLM-S. PBLM-S and PBLM-LS have higher average work times per thread than PBLM-L, which is an expected result since threads will always place new boxes in shared queues, which will potentially minimize the average iddle time per thread. However, we observe that a more careful choice of which box to process next yields better average total running times overall, even when the average work time per thread decreases. As expected, larger values for N_B in PBLM-L significantly decrease the average work time per thread, given that new boxes will become available in the shared queue at a much slower rate. PBLM-LS shows the benefit of having a collection of shared queues, as not only we see an increase in the average work time per thread when compared to PBLM-L, but also in average total time given that solvers now benefit from the information obtained in models previously solved to speed up the optimization models in the next iterations. With 20 threads, PBLM-LS achieves an average speed-up factor of 18.86 when compared to the sequential implementation of BLM.

2.7 Final remarks

The algorithms presented and analyzed in this chapter show that great progress has been made in solving bi-objective mixed integer programs. Consequently, the next major challenge is the development of algorithms for tri-objective mixed integer linear programs, of which, at the time of the writing of this thesis, there are none.

Table 2.8: Avg Run times of 10 experiments for 21dat running the different variants of PBLM.

Instance	N_{BLM}	N_S	**Algorithm**	**Avg. TT**	**Avg. Speed-Up**	**Avg. Work(%)**
21dat	5	4	PBLM-S	1300.27	-	99.69
			PBLM-L ($N_B = 2$)	1235.37	1.05	99.45
			PBLM-L ($N_B = 4$)	1221.13	1.06	99.11
			PBLM-L ($N_B = 8$)	1263.88	1.02	98.01
			PBLM-LS	1199.66	1.08	99.69
	10	2	PBLM-S	691.42	-	99.02
			PBLM-L ($N_B = 2$)	675.11	1.02	98.10
			PBLM-L ($N_B = 4$)	658.82	1.05	97.23
			PBLM-L ($N_B = 8$)	678.55	1.01	96.06
			PBLM-LS	626.16	1.10	98.30
	20	1	PBLM-S	461.99	-	97.27
			PBLM-L ($N_B = 2$)	442.18	1.04	95.65
			PBLM-L ($N_B = 4$)	457.64	1.00	92.22
			PBLM-L ($N_B = 8$)	485.64	0.91	88.80
			PBLM-LS	413.34	1.11	96.97

CHAPTER 3
HEURISTICS FOR THE SAME-DAY DELIVERY PROBLEM WITH HUB CAPACITY CONSTRAINTS

3.1 Introduction

We study a new service network design problem for an urban same-day delivery system in which the number of vehicles that can simultaneously load or unload at a hub is limited. These hub capacity constraints can cause waiting at hubs, which, in turn, can result in certain shipment paths no longer being time feasible, especially in urban same-day delivery environments with aggressive service offerings. In long-haul transportation systems of consolidation carriers, such hub capacity constraints are usually not relevant, because the hubs are large (some very large) with many loading and unloading docks. However, in short-haul transportation systems in urban areas, such hub capacity constraints are critical, because real-estate is expensive (and will continue to become more expensive) and the hubs are small (some very small) with only a few loading and unloading docks. Although service network design models and methods have been developed for various settings, to the best of our knowledge, none of these have considered accommodating these hub capacity constraints, especially not in a same-day delivery environment.

The restriction on the number of vehicles that can simultaneously be loaded and unloaded at a hub has an important consequence, which is typically not encountered in service network design problems: the existence of a feasible solution is no longer guaranteed. For example, it may not be feasible to send all packages directly from their origin to their destination as this requires a large number of vehicles and the limit on the number of vehicles that can be loaded and unloaded simultaneously at hubs may result in so much waiting that some packages will no longer be able to meet their service guarantee. Therefore, this

service network design variant has two objectives: (1) maximize the number of origin-destination markets that can be served, and (2) given the origin-destination markets that are served, minimize the cost of serving these markets.

The variant of the service network design problem investigated is motivated by the settings encountered in China's major mega-cities at package express carrier SF Express (and the instances used in our computational study are representative of these settings). The same-day service products of interest include the "12-18 service", the "14-20 service", and the "12-20 service". Each service product specifies the latest time customers should present their packages to the carrier and the latest time that the carrier guarantees delivery takes place. For example, a customer choosing the "12-18 service" product has to present their packages to the carrier before noon, and the carrier guarantees that these packages will be delivered no later than 6 PM (the same day). This does not mean that all packages for service product *12-18* are available at an origin hub at noon, because in the SF Express system, couriers collect packages from customers and deliver these packages to an *access point* (a parcel box), and then riders deliver packages from the access point to the origin hub. Similarly, it does not mean that all packages for service product *12-18* need to be at the destination hub by 6 PM, because riders need to deliver the packages to an access point, where they are picked up by couriers to handle the ultimate delivery. Therefore, a certain amount of time (1.5 to 2 hours) has to be reserved for these "pre-processing" and "post-processing" activities when designing the same-day service network. For example, if 2 hours are reserved for the pre-processing and post-processing activities for service product *12-18*, then the carrier needs to transport the package from its origin hub to its destination hub between 2 PM and 4 PM.

The service network design problem with hub capacity constraints that we consider seeks for every market, i.e., for every origin – destination pair of hubs, a path specifying how the packages in this market will be transported from their origin to their destination, and a minimum cost vehicle schedule for transporting the packages that specifies for each

vehicle movement, its origin and destination, when loading at the origin starts (and ends), when it departs from the origin (and when it arrives at the destination), and when unloading at the destination starts (and ends), while ensuring that service constraints are satisfied, i.e., loading of packages at their origin starts after they become available and unloading of packages at their destination completes before they are due, and satisfies the operating restrictions at hubs, i.e., the restrictions on the number of vehicles that can simultaneously be loaded and unloaded.

3.1.1 Contributions

We model this service network design problem using a time-expanded network in which arcs represent the loading, travel, waiting, and unloading of a vehicle and commodities represent packages in a market, and show how this model can be formulated as an integer program (IP). Since the IP cannot be solved in an acceptable amount of time for real-world instances, we develop two heuristic algorithms for obtaining good-quality solutions: (1) a metaheuristic, and (2) a hybrid matheuristic, which takes advantage of the strengths of the metaheuristic and from an IP based heuristic proposed in [41]. Computational experiments show that the hybrid matheuristic produces high-quality solutions in a reasonable amount of time. The contributions of this research are summarized as follows:

- A new service network design variant is introduced for same-day delivery systems in urban areas and a non-trivial integer programming model for its solution is developed;
- A metaheuristic and a hybrid matheuristic, which combines the strengths of the IP-based heuristic and the metaheuristic, are designed and implemented; and
- The efficacy of the heuristics is demonstrated on a number of real-world instances.

The remainder of this chapter is organized as follows. In Section 3.2, we review relevant prior research. In Section 3.3, we provide a formal description of the same-day delivery system and present a model and formulation using an appropriately constructed

time-expanded network. In Section 3.4, we introduce two heuristics: a metaheuristic, and a hybrid matheuristic, which takes advantage of the strengths of the metaheuristic and from an IP based heuristic approach. In Section 3.5, we present and interpret the results of an extensive computational study. Finally, in Section 3.6, we finish with some final remarks.

3.2 Literature Review

Since we are not aware of any literature addressing the design of a service network for same-day delivery of packages with loading and unloading capacities at hubs, we briefly review literature on same-day delivery, on service network design for package express carriers, on service network design with other types of capacity limitations, and on advances in solving service network design problems.

There is a growing body of literature on same-day delivery, but it is focused almost exclusively on the delivery of packages from a fulfillment center, which gives rise to challenging dynamic routing and scheduling problems (see, e.g., [42], [43], [44], and [45]), but is quite different from the same-day delivery environment studied in this thesis, which is characterized by package flows between hubs. There is literature on courier operations and dial-a-ride services in urban areas, which do involve taking packages or passengers from a pickup location to a drop-off location (see, e.g., [46], [47], and [48]), but consolidation and transfers are usually not as critical to feasibility and profitability as they are in the same-day delivery environment under concern.

There is a fair amount of literature on optimizing the design and operations of package express carriers service networks, but the focus has been on inter-city package flows rather than same-day delivery of packages within a city. Examples include [9], [49], [50], and [51].

Researchers have incorporated capacity limitations into service network design models in a variety of transportation systems. In the context of an air transportation system, [52] proposes a multi-commodity minimum-cost flow model on a time-expanded network which

incorporates the arrival capacity at destination airport of flights. In the context of a maritime transportation system, where the number of available berths in a port strongly impacts the throughput capacity of a port, [53] formulate the dynamic berth allocation problem as an integer multi-commodity network flow problem, using innovative flexible berth-space utilization scheme based on blocking plans. In rail networks, the capacity of a classification yard limits the formation of blocks, i.e., sets of railcars that travel together on part of their journey from origin to destination. [54] formulate the blocking problem, i.e., the creation of cost effective blocks, as a network design problem, with nodes and arcs representing yards and candidate blocks, respectively. Limited yard capacity is incorporated by imposing maximum in- and outdegrees on the nodes. In the context of service network design for intercity package flows, [51] consider the (limited) sorting capacity at terminals.

Recent research advances in exact solution methods for solving service network design problems have come from iterative refinement techniques ([55], [56]) and branch-and-price-and-cut algorithms ([57], [58]). More effective matheuristics, i.e., approaches integrating metaheuristic concepts and mathematical programming techniques, continue to appear. [59] present a matheuristic that iterates between linear programming and slope scaling for multicommodity capacitated fixed-charge network design. [60] combine column generation, metaheuristic, and exact optimization techniques in dealing with service network design with resource constraints. In liner shipping network design, [61] use an integer program to search for improvements in their matheuristic. For less-than-truckload load plan design, [62] develop an effective neighborhood search heuristic for solving a natural integer programming model where a modified and restricted version of the integer program is solved to find improving changes during each iteration of the matheuristic.

3.3 Problem Statement

Let $D = (N, A)$ be a flat network with node set N modeling physical locations or hubs and directed arc set A modeling travel between locations. A travel time $\tau_{ij} \in \mathbb{N}_+$ and a

travel cost $c_{ij} \in \mathbb{R}_+$ are associated with each arc $a = (i, j) \in A$. Let K denote a set of commodities to be served, each of which has a single source node $o_k \in N$ (later referred to as the commodity's origin), a single sink node $d_k \in N$ (later referred to as the commodity's destination), and a quantity $q_k \in \mathbb{R}_+$ that needs to be routed from its origin to its destination along a single geographic path $P_k = (o_k, \ldots, d_k)$ using a homogeneous fleet of vehicles with capacity $Q \in \mathbb{N}_+$. We assume that $q_k \leq Q$ for all $k \in K$. Commodity $k \in K$ becomes available at its origin at time $e_k \in \mathbb{N}_+$ and is due at its destination at time $l_k \in \mathbb{N}_+$. We assume that each commodity $k \in K$ will be loaded every time it departs from a hub $i \in N$ and will be unloaded every time it arrives at a hub $i \in N$. This assumption is in line with practices at many package express carriers. It simplifies operations at hubs; selective loading and unloading packages is prone to human error, especially under time pressure. We assume the loading of vehicle takes $\tilde{\tau}_l \in \mathbb{N}_+$ and the unloading of a vehicle takes $\tilde{\tau}_u \in \mathbb{N}_+$. For ease of presentation, we assume these times are the same for all hubs, but it is easy to accommodate hub-dependent loading and unloading times. Each hub $i \in N$ has loading capacity $L_i \in \mathbb{N}_+$ and an unloading capacity $U_i \in \mathbb{N}_+$, which represent the maximum number of vehicles that can be loaded and unloaded at the same time, respectively. We assume that there is enough space at each hub for vehicles to wait if loading or unloading cannot start upon arrival. Finally, we assume, without loss of generality, that a vehicle departs as soon as it is loaded. Thus, a vehicle may only wait before it gets loaded or before it gets unloaded.

The Service Network Design with Hub Capacities (SNDHC) problem seeks to determine a path P_k for each commodity $k \in K$ and a vehicle schedule that implies loading and unloading start times at each hub in the path P_k, such that the loading at the origin hub starts at or after e_k and the unloading at the destination hub starts at or before $l_k - \tilde{\tau}_u$ and that satisfies the hub capacity constraints, i.e., for every hub $i \in N$ and at any time $t \in [\min_{k \in K} e_k, \max_{k \in K} l_k]$, there are no more than L_i vehicles loading and no more than U_i vehicles unloading. The restriction on the number of vehicles that can simultaneously

be loaded and unloaded at a hub implies that the existence of a feasible solution is no longer guaranteed. Therefore, the objective of SNDHC is to minimize the cost of a vehicle schedule that maximizes the number of commodities for which a feasible path can be found.

We use a time-expanded network to model SNDHC. We derive a time-expanded network $\mathcal{D} = (\mathcal{N}, \mathcal{A} \cup \mathcal{H})$ from flat network D and a set of time points $T = \bigcup_{i \in N} T_i$ with $T_i = \{t_1^i, \ldots, t_{n_i}^i\}$. The timed node set $\mathcal{N}$ has a node (i, t) for each node $i \in N$ and $t \in T_i$. The holding arc set $\mathcal{H}$ contains arcs $((i, t_g^i), (i, t_{g+1}^i))$ for all $i \in N$ and $g = 1, \ldots, n_i - 1$. A holding arc $((i, t_g^i), (i, t_{g+1}^i))$ models the possibility of holding packages at hub i for a period of time while they wait to be loaded onto a vehicle. The movement arc set $\mathcal{A}$ contains arcs of the form $((i, t), (j, \bar{t}))$, where $(i, j) \in A$, $t \in T_i$, and $\bar{t} \in T_j$. An arc $((i, t), (j, \bar{t}))$ models the possibility of sending packages from hub i to hub j with the loading of packages starting at i at time t and the unloading of packages finishing at j at time $\bar{t}$. This implies that the vehicle departs from i at time $t + \tilde{\tau}_l$ and waits at j from $t + \tilde{\tau}_l + \tau_{ij}$ until $\bar{t} - \tilde{\tau}_u$. (Thus, we must have that $(\bar{t} - \tilde{\tau}_u) - (t + \tilde{\tau}_l) \geq \tau_{ij}$.)

Observe that arc $((i, t), (j, \bar{t}))$ and arc $((i, t), (j, \bar{t}'))$ represent a different set of activities even though the vehicle travels at the exact same time (departing at i at $t + \tilde{\tau}_l$ and arriving at j at $t + \tilde{\tau}_l + \tau_{ij}$). This differs from most service network design problems, where it suffices to model vehicle movements, i.e., have arcs of the form $((i, t), (j, t + \tau_{ij}))$. However, to be able to accurately model the hub capacity constraints, it is necessary to explicitly embed the loading and unloading of a vehicle into the arc. Different packages traveling from i to j on their journey from their origin to their destination can start loading at i at the same time t but start unloading at j at different times $\bar{t} - \tilde{\tau}_u$ at $\bar{t}' - \tilde{\tau}_u$ due to differences in waiting time at j of the vehicle that transports them.

We use a time-expanded network $\mathcal{D}^\Delta$ derived from $\mathcal{D}$ with a regular time discretization controlled by parameter $\Delta \in \mathbb{N}_+$. Specifically, we let $T_i = \{E\Delta, (E+1)\Delta, \cdots, L\Delta\}$ for all $i \in N$ where $E, L \in \mathbb{N}_+$ with $\min_{k \in K} e_k/\Delta - 1 < E \leq \min_{k \in K} e_k/\Delta$ and

$\max_{k\in K} l_k/\Delta \leq L < \max_{k\in K} l_k/\Delta + 1$. In the time-expanded network $\mathcal{D}^\Delta$, for every pair of nodes (i,t) and $(j,\bar{t})$ where $(i,j) \in A$ and $(\bar{t} - \tilde{\tau}_u) - (t + \tilde{\tau}_l) \geq \tau_{ij}$, there is a movement arc $((i,t),(j,\bar{t}))$ in $\mathcal{A}$, i.e., we consider all possible loading and unloading time options for every possible vehicle travel option. To handle the discretization of time, we adopt a standard mapping that rounds up travel times, rounds up loading and unloading times, rounds up commodity availability times, and rounds down times commodity due times, so that a feasible solution to the service network design model on the time-expanded network $\mathcal{D}^\Delta$ can always be converted to a feasible schedule in continuous time.

We denote the service network design problem with hub capacities defined on the time-expanded network $\mathcal{D}^\Delta$ described above by SNDHC($\mathcal{D}$). Let $y_{ij}^{t\bar{t}}$ represent the number of times arc (i,j) is used to accommodate dispatches that start loading at hub i at time t and finish unloading at hub j at time $\bar{t}$. Because multiple vehicles can perform the same movement, the $y_{ij}^{t\bar{t}}$ variables have to be integer. Let $x_{ij}^{kt\bar{t}}$ represent whether commodity $k \in \mathcal{K}$ travels on movement arc $((i,t),(j,\bar{t}))$. For convenience, let $\mathcal{A}^k$ and $\mathcal{H}^k$ denote the movement and holding arc sets containing movement and holding arcs that can possibly be used by commodity $k \in K$, respectively. Let $\mathcal{N}'$ be a timed node set that contains a node (i,t) for each node $i \in N$ and $t \in T_i'$ where $T_i' = \{\min_{k\in K} e_k, \min_{k\in K} e_k + 1, \ldots, \max_{k\in K} l_k\}$. Because each commodity must follow a single path from its origin to its destination, the $x_{ij}^{kt\bar{t}}$ variables have to be binary. Finally, let z_k be a binary variable indicating whether commodity $k \in K$ is served or not. With these variables, SNDHC($\mathcal{D}$) can be formulated as follows:

$$\max \quad \sum_{k\in K} z_k \qquad \text{(Phase 1)}$$

$$\min \quad \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}} c_{ij} y_{ij}^{t\bar{t}} \qquad \text{(Phase 2)}$$

s.t.

$$\sum_{((i,t),(j,\bar{t}))\in\mathcal{H}^k\cup\mathcal{A}^k} x_{ij}^{kt\bar{t}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{H}^k\cup\mathcal{A}^k} x_{ji}^{k\bar{t}t} = \begin{cases} +z_k & (i,t) = (o_k, e_k), \\ -z_k & (i,t) = (d_k, l_k), \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, (i,t) \in \mathcal{N} \tag{3.1a}$$

$$\sum_{k\in K} q_k x_{ij}^{kt\bar{t}} \leq Q y_{ij}^{t\bar{t}} \qquad \forall((i,t),(j,\bar{t})) \in \mathcal{A} \tag{3.1b}$$

$$\sum_{((i,s),(j,\bar{s}))\in\mathcal{A}: t-\tilde{\tau}_l < s \leq t} y_{ij}^{s\bar{s}} \leq L_i \qquad \forall(i,t) \in \mathcal{N}' \tag{3.1c}$$

$$\sum_{((i,s),(j,\bar{s}))\in\mathcal{A}: \bar{t} \leq \bar{s} < \bar{t}+\tilde{\tau}_u} y_{ij}^{s\bar{s}} \leq U_j \qquad \forall(j,\bar{t}) \in \mathcal{N}' \tag{3.1d}$$

$$x_{ij}^{kt\bar{t}} \in \{0,1\} \qquad \forall((i,t),(j,\bar{t})) \in \mathcal{A}^k \cup \mathcal{H}^k \quad \forall k \in K \tag{3.1e}$$

$$y_{ij}^{t\bar{t}} \in \mathbb{N}_{\geq 0} \qquad \forall((i,t),(j,\bar{t})) \in \mathcal{A} \tag{3.1f}$$

That is, SNDHC($\mathcal{D}$) problem seeks to maximize the total number of commodities to be served on time in Phase 1, and to minimize the total travel cost in Phase 2 while ensuring the maximum number of served commodities. Constraints (3.1a) ensure that each served commodity departs from its origin after it becomes available and arrives at its destination before it is due. The presence of holding arcs allows a commodity to arrive early at its destination or depart late from its origin. Constraints (3.1b) ensure that a sufficient number of vehicles is available for the commodities that are sent from hub i to hub j starting loading at time t and completing unloading at time $\bar{t}$. Constraints (3.1c) and (3.1d) ensure that the number of vehicles that are loading or unloading simultaneously does not exceed the specified hub capacity limits across all locations at any time during the planning horizon. Constraints (3.1e) and (3.1f) define the variables and their domains.

Since there is a movement arc $((i,t),(j,\bar{t})) \in \mathcal{A}$ for every pair of nodes (i,t) and $(j,\bar{t})$ with $(i,j) \in A$ and $(\bar{t} - w_u) - (t + w_l) \geq \tau_{ij}$, the number of variables $x_{ij}^{kt\bar{t}}$ and $y_{ij}^{t\bar{t}}$ is huge; prohibitively large for real-life instances. Furthermore, to ensure the hub capacity

restrictions across all locations and at any time of the planning horizon, the number of loading and unloading capacity constraints is huge too; prohibitively large for real-life instances. Therefore, solving SNDHC($\mathcal{D}$), or even obtaining high-quality solutions, using a standard commercial solver is practically impossible, which motivates the development of heuristics.

3.4 Methodology

3.4.1 Metaheuristic Approach

We start by discussing a multi-start iterated local search (MILS) metaheuristic for solving instances of SNDHC($\mathcal{D}$). Its primary benefit is that it can quickly produce high quality solutions. The shorter solve time allows MILS to be used for sensitivity analysis, as will be discussed in Section 3.5.2.

MILS focuses on maximizing the number of commodities served and only uses cost information for breaking ties. MILS has two main loops: an *outer* loop and an *inner* loop; see Algorithm 1. In the outer loop, we perform It^O_{MAX} iterations where, in each iteration, we start from a greedy initial solution $\mathcal{S}$ and perform an iterated neighborhood search to improve it, always keeping track of the best solution found during the execution of the algorithm. After an initial solution has been constructed, an attempt is made to improve that solution by iteratively examining pairs of hubs i and j that either have at least one timed path that connects them, or for which there is a commodity originating at i and destined for j that is not served in $\mathcal{S}$. The inner loop terminates after It^I_{max} consecutive iterations without an improvement.

The loading and unloading operations at a hub can be viewed as a sequence of time periods, each with a given start time. For implementation efficiency, such a sequence of time periods is stored in a binary search tree, as illustrated in Figure 3.1. The binary search trees allows us to quickly detect conflicts between any loading or unloading operations at a hub, which is critical to ensure that the hub capacity constraints are respected. More

Algorithm 1 Multi-Start Iterated Local Search

input: The sets of hubs and commodities and the travel time matrix
output: A set of paths $\mathcal{S}^*$ serving a large number of commodities
$\mathcal{S}^* \leftarrow \{\}$
$It_{\text{MS}} \leftarrow 0$
while $It_{\text{MS}} < It^{O}_{\text{MAX}}$ **do** # Outer loop
  $\mathcal{S} \leftarrow$ INITIALSOLUTION()
  $\{P, R\} \leftarrow$ RESETPROBABILITIES()
  while termination criteria is not satisfied **do** # Inner loop
    for all pairs of hubs (i,j) in random order **do**
      $\mathcal{S} \leftarrow$ NEIGHBORHOODSEARCH(i, j, $\mathcal{S}$, P, R)
      $\mathcal{S} \leftarrow$ REMOVEINFEASIBLEPATHS($\mathcal{S}$)
      $\mathcal{S} \leftarrow$ CONNECTPATHS($\mathcal{S}$)
      if $c_1(\mathcal{S}) > c_1(\mathcal{S}^*)$ **then**
        $\mathcal{S}^* \leftarrow \mathcal{S}$
      end if
    end for
    $P \leftarrow$ UPDATEPROBABILITIES(P, R)
  end while
  $It_{\text{MS}} \leftarrow It_{\text{MS}} + 1$
end while
return $\mathcal{S}^*$

specifically, determining whether a given loading or unloading operation (a time period with a given start time) can be added at a hub can be done in $\mathcal{O}(log\ w)$ time, where w is the number of loading or unloading operations in the search tree. Updating the search tree, i.e., adding or deleting a loading or unloading operation, can also be done in $\mathcal{O}(log\ w)$ time.

A solution $\mathcal{S}$ is represented as a set of timed paths and an assignment of commodities to these paths. Each timed path is composed of a sequence of sets of two actions: (1) a vehicle loading period v_i^l (the time period representing when vehicle v starts and finishes loading at location i) and (2) a vehicle unloading period v_j^u (the time period representing when the vehicle v starts and finishes unloading at location j. For example, a timed path $(i \rightarrow j \rightarrow k)$ is represented as the sequence $\{v_i^l, v_j^u, v_j^l, v_k^u\}$, where each element is stored in the binary search tree representing the activities at the loading or unloading dock used by the vehicle.

Next, we will describe how we obtain an initial solution in each iteration of the outer

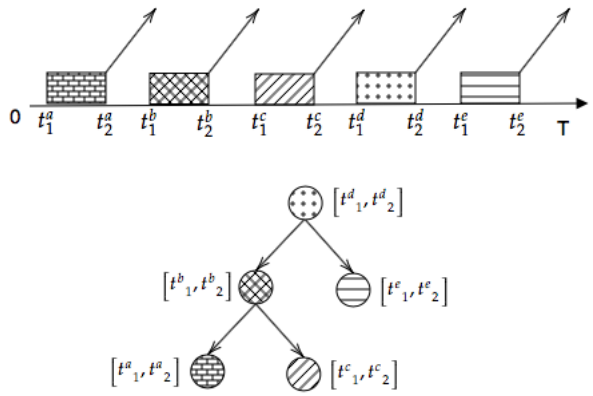


Figure 3.1: Binary Search Tree representing the loading operations over time at a loading dock of a hub

loop, the neighborhoods used in the local search procedure, and how the neighborhoods are selected in the local search procedure.

Initial solution construction.

We start by listing all possible pairs of hubs (i, j) with i representing the origin of a commodity and j representing its destination, respectively, and partition them into three groups, G_1, G_2 and G_3, based on the travel time between the hubs. The travel time is short for pairs in G_1, moderate for pairs in G_2, and long for pairs in G_3. We start constructing the initial solution $\mathcal{S}$ by processing the commodities associated with pairs in G_1 in random order, one by one. Given a pair $(i, j) \in G_1$, we try to create feasible timed paths for all commodities originating at i and destined to j (there can be more than one such commodity because of the different service classes) departing as early as possible from i, always respecting hub capacities and the available time at the origin and the due time at the destination. If a feasible timed path is identified, then $(i \rightarrow j)$ is added to the list of timed paths. Before considering the next pair, we check if the path $(i \rightarrow j)$ enables the creation of additional paths, i.e., if a timed path $(j \rightarrow k)$ exists and the time path $(i \rightarrow j \rightarrow k)$ is feasible, then $(i \rightarrow j \rightarrow k)$ is created as it can potentially serve commodities originating at i and destined to k. Once all pairs in G_1 are processed, we repeat the same procedure for the pairs in G_2

and G_3.

Local search.

MILS employs several neighborhoods to try and improve a given solution, each involving simple modifications of timed paths or of loading and unloading sequences. The exploration of the neighborhoods is exhaustive, i.e., the solution modifications or moves are performed for all hub pairs (i, j), in random order. Whenever a pair of hubs (i, j) is selected in the neighborhood search, we first check if all commodities originating from i and destined for j are served. In case there is a commodity that is not served, we first try to create a direct timed path from i to j at the earliest possible departure time from i. If no direct timed path is possible, we try to create an indirect timed path ($i \rightarrow k \rightarrow j$), where k is chosen randomly from among the hubs that are within given distance from either i or j. Specifically, we check if there exists a timed path $(k \rightarrow j)$ that arrives at j before the due time of the commodity and if it is possible to send a vehicle from i to k such that it arrives at k in time, i.e., we only consider creating a new timed path from i to k. In case all commodities are served, or, if we are unable to create paths for all the commodities that are not yet served, we apply one of the following neighborhoods:

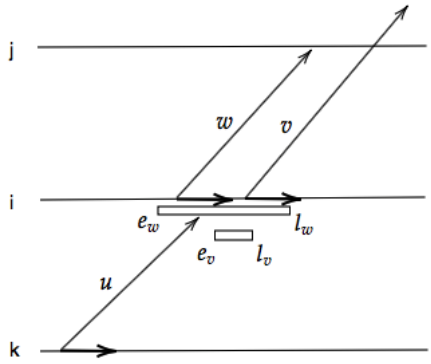
- MOVELOADINGPERIOD(i,j) and MOVEUNLOADINGPERIOD(i,j): Each neighborhood tries to delay or advance (chosen with equal probability) the loading or unloading periods of a timed path $(i \rightarrow j)$ as much as possible. New docks may be assigned at either i or j, if beneficial.
- CHANGELOADINGDOCK(i,j) and CHANGEUNLOADINGDOCK(i,j): Each neighborhood tries to change the loading or unloading dock of a timed path $(i \rightarrow j)$ in order to create space at the dock to add new loading or unloading operations (which can then be used for other timed paths). The start time of the loading or unloading operation is kept the same.

– SWITCHLOADINGPERIOD(i,j): The neighborhood tries to switch the loading operation of a direct timed path $(i \rightarrow j)$ with the loading operation of another, randomly selected, direct timed path $(i \rightarrow k)$ at the loading dock. New unloading times and docks may be assigned to the vehicles at both j and k, if necessary. The switch is only performed if it does not affect the feasibility of the commodities assigned to the timed paths.

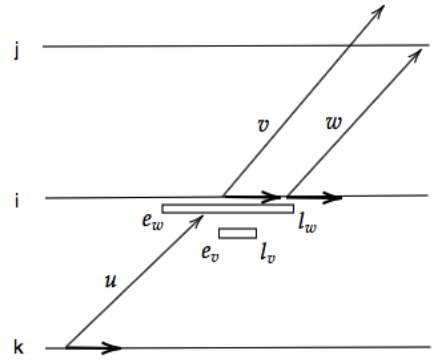
– CROSSOVERPATHS(i,j): Let $(i \rightarrow k \rightarrow j)$ and $(l \rightarrow m \rightarrow j)$ be two distinct timed paths in $\mathcal{S}$. The neighborhood generates new paths by crossing over $(i \rightarrow k \rightarrow j)$ and $(l \rightarrow m \rightarrow j)$ into two new paths: $(i \rightarrow m \rightarrow j)$ and $(l \rightarrow k \rightarrow j)$. The crossover operation is only applied if the new paths are feasible and do not violate the loading/unloading capacity constraints at the related hubs.

– SWITCHDIRECTPATH(i,j): The neighborhood tries to reassign the commodities on the direct timed path $(i \rightarrow j)$ to an indirect timed path $(i \rightarrow k \rightarrow j)$, where k is randomly selected. If all commodities can (feasibly) be reassigned, then $(i \rightarrow j)$ is removed from $\mathcal{S}$, hence opening space for a new loading operation at hub i and a new unloading at hub j.

– RE-ARRANGELOADINGPERIODS(i): Consider the hubs i, j and k and the movement arcs u, v, and w shown in Figure 3.2a. Each movement arc reflects the time that the loading starts and the time that the unloading finishes. The accompanying rectangles represent the feasible times at which the loading can start, which are determined by the latest time that one of the commodities transported becomes available at i and the departure time at i that ensures an arrival at the time unloading starts. Next, assume that there is a commodity that originates at k and is destined for j. In this example, it is possible to create an indirect timed path $(k \rightarrow i \rightarrow j)$ by starting the loading of v earlier and starting the loading of w later (but the start times remain within their respective windows, as shown in Figure 3.2b.



(a) Initial state of the loading operations (large rectangles) w,v and u in hubs i and k, respectively, and the associated time windows $[e, l]$ where w and v can start loading. In this example, there is an unserved timed commodity path from k to j.



(b) Final state of the loading operations in hub i after the neighborhood move is applied. It is possible to create the indirect timed path $(k \rightarrow i \rightarrow j)$ by starting the loading of v earlier and starting the loading of w later, while still respecting the associated time windows.

Figure 3.2: Initial and final states of the loading operations at a hub after the neighborhood move RE-ARRANGELOADINGPERIODS is applied.

This neighborhood attempts to re-organize the loading operations at the loading dock of hub i by solving a small integer program, which seeks to maximize the number of potential new indirect timed paths for commodities not yet served while ensuring that no existing timed paths become infeasible. For a given movement arc w originating at hub i, let d_w denote the destination, let e_w denote the earliest time that loading of w can start, let l_w denote the latest time that loading of w can start, and let s_w be a variable that represents the time that the loading of w starts. Finally, let there be a commodity that originates at k and is destined for d_w that is not yet served and a vehicle that originates at k and finishes unloading at i at time a_{k,d_w}. We solve the following IP:

$$\max \sum_{k,d_w} \lambda_{k,d_w}$$

s.t

$$s_w + \bar{\tau}_l \leq s_y + M \cdot (1 - \delta_{wy}) \qquad \forall w, y \in \mathcal{W}, w \neq y \qquad (3.2a)$$

$$\delta_{wy} + \delta_{yw} = 1 \qquad \forall w, y \in \mathcal{W} \qquad (3.2b)$$

$$e_w \leq s_w \leq l_w \qquad \forall w \in \mathcal{W} \qquad (3.2c)$$

$$s_w \geq a_{k,d_w} \lambda_{k,d_w} \qquad \forall w \in \mathcal{W}, \forall (k, d_w) \qquad (3.2d)$$

$$\lambda_k \in \{0, 1\} \qquad \forall k \qquad (3.2e)$$

$$\delta_{wy} \in \{0, 1\} \qquad \forall w, y \in \mathcal{W} \qquad (3.2f)$$

$$s_w \in \mathbb{R}_+ \qquad \forall w \in \mathcal{W} \qquad (3.2g)$$

where $\mathcal{W}$ is the set of movement arcs originating at i, λ_{k,d_w} is a binary variable indicating whether or not a new indirect timed path can be created to served a commodity originating at k and destined for d_w, δ_{wy} is a binary variable indicating whether or not the start of the loading of w precedes the start of the loading of y, and M is a large constant. Constraints (3.2a) and (3.2b) ensure that there are no overlaps be-

tween loading periods, while constraint (3.2c) ensure that no existing paths become infeasible, and constraint (3.2d) links the decision variables λ_{k,d_w} and s_w. Finally, constraints (3.2e),(3.2f) and (3.2g) define the decision variables and their domains.

– RE-ARRANGEUNLOADINGPERIODS(i): This neighborhood is similar to the previous one, but now focused on re-arranging the unloading operations at an unloading dock of hub i.

– OPENNEWLOADINGPERIOD(i): The neighborhood tries to re-arrange the loading operations at a loading dock of hub i, again using a small integer program, now with the goal of opening space for a new loading operation at the dock. The integer program is similar to the one presented in RE-ARRANGELOADINGPERIODS. It has an additional decision variable z representing the start time of the new loading operation and additional constraints that link the variable z to the latest possible time of the start of a loading operation for a commodity that is not yet served. We solve the following IP:

$$\max \sum_k \gamma_k$$

s.t

$$s_w + \bar{\tau}_l \leq s_y + M \cdot (1 - \delta_{wy}) \qquad \forall w, y \in \mathcal{W}, w \neq y \qquad (3.3a)$$

$$z + \bar{\tau}_l \leq s_w + M \cdot (1 - \delta_{zw}) \qquad \forall w \in \mathcal{W} \qquad (3.3b)$$

$$s_w + \bar{\tau}_l \leq z + M \cdot (1 - \delta_{wz}) \qquad \forall w \in \mathcal{W} \qquad (3.3c)$$

$$\delta_{wy} + \delta_{yw} = 1 \qquad \forall w, y \in \mathcal{W} \cup \{z\} \qquad (3.3d)$$

$$e_w \leq s_w \leq l_w \qquad \forall w \in \mathcal{W} \qquad (3.3e)$$

$$e_k \lambda_k \leq z \leq l_k - M \cdot (1 - \beta_k) \qquad \forall k \qquad (3.3f)$$

$$\lambda_k + \beta_k = 2\gamma_k \qquad \forall k \qquad (3.3g)$$

$$\lambda_k \in \{0, 1\} \qquad \forall k \qquad (3.3h)$$

$$\beta_k \in \{0,1\} \qquad \forall k \tag{3.3i}$$

$$\gamma_k \in \{0,1\} \qquad \forall k \tag{3.3j}$$

$$\delta_{wy} \in \{0,1\} \qquad \forall w, y \in \mathcal{W} \cup \{z\} \tag{3.3k}$$

$$s_w \in \mathbb{R}_+ \qquad \forall w \in \mathcal{W} \tag{3.3l}$$

$$z \in \mathbb{R}_+ \tag{3.3m}$$

where e_k is the earliest time that a vehicle can start loading at the hub in order to carry commodity k, l_k is latest time that a vehicle can start loading at hub i and finish unloading at the destination of k before its due time, α_k and β_k are auxiliary binary decision variables and γ_k is a binary decision variable indicating whether or not a new indirect timed path can be created to serve commodity k. If new indirect timed paths are possible for more than one such commodity, a single new direct timed path is chosen randomly with equal probability.

– OPENNEWUNLOADINGPERIOD(i): This neighborhood is similar to the previous one, but now focused on opening space for a new unloading operation at an unloading dock of hub i.

As the neighborhood moves, which are designed to find timed paths for commodities that currently do not yet have a timed path, may also result in destruction of existing timed paths before carrying out a move, we examine the difference between the number of commodities with a timed path before and after the the move is performed. If the number of commodities with a timed path remains the same or increases, the move is performed, but if the number of commodities with a timed path decreases, but not by more than a threshold T_k, it is carried out with probability p. Finally, after performing a move, we first remove the indirect timed paths that have become infeasible, using REMOVEINFEASIBLEPATHS($\mathcal{S}$), and then try to create new indirect timed paths, using CONNECTPATHS($\mathcal{S}$).

Neighborhood selection.

The probability that a neighborhood is selected depends on how many timed paths are added/removed by the neighborhood throughout the execution of the metaheuristic. The more successful a particular neighborhood is in creating new timed paths, the higher the chance this neighborhood will be chosen in subsequent iterations. Let P_n be the current probability of selecting neighborhood n and let R_n be the current reward associated to n, i.e, the difference between the total number of timed paths added and removed when using neighborhood n in past iterations. When UPDATEPROBABILITIES(P,R) is called, the selection probabilities associated with the neighborhoods are updated, as shown in Algorithm 2, where N is the number of neighborhoods and η is a parameter determining how aggressively we increase or decrease the probabilities based on the rewards vector. Before entering the inner loop, RESETPROBABILITIES() resets all probabilities and rewards by setting $\{P_n = \frac{1}{N}, R_n = 0\}\ \forall n \in N$. In order to maintain an effective level of diversification, we reset all probabilities to the discrete uniform distribution $\mathcal{U}(1, N)$ if no improvements were made to the current solution for It_R^I consecutive iterations.

Algorithm 2 Update neighborhood selection probabilities

input: The probability and the rewards vectors P and R
output: The updated probability vector P.
$\bar{P} \leftarrow P$
$W \leftarrow 0$
for $n = 1, \cdots, N$ **do**
    $\bar{P}_n = P_n \cdot e^{\eta R_n}$
    $W \leftarrow W + \bar{P}_n$
end for
for $n = 1, \cdots, N$ **do**
    $\bar{P}_n = \frac{\bar{P}_n}{W}$
end for
return $\bar{P}$

3.4.2 Hybrid Matheuristic

The IP-based heuristic (IP-H) proposed in [41] solves multiple small hierarchical IPs derived from the original IP presented in Section 3.3, which greatly reduces the computational effort required for finding high quality solutions. This heuristic approach solves one hierarchical IP for every hub that serves as origin or destination for at least one commodity in a pre-specified sequence, where the goal is twofold: (1) maximize the number of commodities to be served that originate from or are destined to that hub and (2) minimize the cost of the timed feasible paths assigned for such commodities. When processing a specific hub, commodities originating from or destined to that hub are allowed to follow any feasible timed path in the network, but commodities which have been assigned a feasible path in previous processed hubs in the sequence are forced to follow either the previously assigned path or a direct path from the commodity's origin to its destination.

Given that this IP-heuristic processes hubs in a given sequence, it is obvious that this sequence has an impact on the quality of the resulting solution. Therefore, in this section, we propose a hybrid matheuristic (H-MAT), in which solutions produced by the metaheuristic are used to guide the IP-heuristic. More specifically, the solutions produced by the metaheuristic are used to (1) adjust the sequence in which the hubs are processed, and (2) adjust the (geographic) paths options considered for commodities when a hub is processed.

To control the solution time, we divide the processing of hubs in stages. There will be G stages and each stage involves the processing of a certain number of hubs. The "staging" can be represented by a vector $(m_1, m_2, \ldots, m_G)$ with $m_g > 0$ and $\sum_{g=1}^{G} m_g = |N|$, where m_g represents the number of hubs processed in Stage g. In Stage g, we start by running MILS, where we enforce that commodities that are served in the latest solution (obtained when processing the last hub in Stage $g-1$) have to remain served, and recording the commodities that are served in its solution. Then, the hubs are sorted in nondecreasing order of the number of commodities served that originate from or are destined to the hub, and the first m_g as-yet unprocessed hubs are selected to be processed (one by one).

Let $K_h^1 \subseteq K$ denote the set of commodities for which h is either the origin or the destination, i.e., $K_h^1 = \bigcup_{k \in K: o_k = h} k \cup \bigcup_{k \in K: d_k = h} k$. Let $K_h^2 \subseteq K \backslash K_h^1$ denote the set of commodities (for which h is not the origin or the destination) which have been assigned a feasible path when the previous hub in the sequence was considered (if h is the first hub in the sequence, then $K_h^2 = \emptyset$). The set of commodities considered for hub h is $K_h = K_h^1 \cup K_h^2$. When processing hub h, each commodity $k \in K_h^1$ is allowed to follow any feasible timed path, and each commodity $k \in K_h^2$ is forced to follow either (1) the direct path from the commodity's origin o_k to its destination d_k, or (2) the previously assigned geographic path $P^{k'}$, or (3) the geographic path in the MILS solution. The arc sets $\mathcal{A}^{hk}$ and $\mathcal{H}^{hk}$ for $k \in \mathcal{K}_h$ and $\mathcal{A}^h$ and $\mathcal{H}^h$ are defined as before. Algorithm 3 shows the pseudo-code for H-MAT.

Algorithm 3 Hybrid Matheuristic

input: The sets of hubs, commodities, the travel time/distances between hubs, the number of stages $|G|$ and parameter vector M
output: The set of timed paths $\mathcal{S}^*$
while not all hubs $i \in N$ have been processed **do**
    Run Algorithm 1 while making sure that previously served commodities remain served, and record served commodities and their assigned geographic paths in the updated best MILS result
    Sort hubs based on the number of served commodities that origin from or destine to the hub in the updated best MILS result (ascending)
    Choose the first $m_g \in M$ unprocessed hubs in the sort as the next $m_g \in M$ hubs in the hub sequence N to process
    for `hub` h in the new chosen m_g hubs **do**
        identify commodity set K_h
        identify integrated arc set $\mathcal{A}^{hk}$ and holding arc set $\mathcal{H}^{hk}$ for every $k \in K_h$
        solve the arc-based hierarchical model
    end for
    $g = g + 1$
end while
return $\mathcal{S}^*$

The advantage of H-MAT over IP-H is that the sequence in which the hubs are processed is determined dynamically and informed by the solution produced by MILS and that for commodities $k \in K_h^2$ another geographic path is considered (the geographic path it

uses in the MILS solution).

3.5 Computational Study

3.5.1 Instances

The proposed algorithms were used to solve real-world instances of SF Express, each representing same-day delivery service offerings in one of China's mega-cities. A representative day was used to define the demand to be served by a fleet of homogeneous vehicles, each with a capacity of 400 packages. The demand consists of a set of commodities, each specifying an origin hub, a destination hub, a number of packages, the time the packages are available at the origin hub, and the time the packages are due at the destination hub. Packages that have the same origin and destination hub, but that become available at different times or that have different due times are considered to be different commodities; this happens when more than one service product is offered in a market (e.g., "12-18 service" and "14-20 service"). For each hub, a limit on the maximum number of vehicles that can be simultaneously loaded and that can be simultaneously unloaded are given. A commodity will be loaded every time it departs from a hub and unloaded every time it arrives at a hub; each loading or unloading operation takes 10 minutes. Tables 3.1 and 3.2 summarize the characteristics of the four instances. For each instance, the demand information includes the number of commodities to be served ($|\boldsymbol{K}|$), the length (hours) of the planning period ($|\boldsymbol{T}|$), i.e., $|\max_{k\in K} l_k - \min_{k\in K} e_k|$, the average and standard deviation of the number of packages per commodity ($\bar{\boldsymbol{q}}_{\boldsymbol{k}}$, $\tilde{\boldsymbol{q}}_{\boldsymbol{k}}$), the average and standard deviation of the time that commodities become available ($\bar{\boldsymbol{e}}_{\boldsymbol{k}}$, $\tilde{\boldsymbol{e}}_{\boldsymbol{k}}$), the average and standard deviation of the time commodities are due ($\bar{\boldsymbol{l}}_{\boldsymbol{k}}$, $\tilde{\boldsymbol{l}}_{\boldsymbol{k}}$), the average and standard deviation of the direct travel time (minutes) for the commodities ($\bar{\boldsymbol{\tau}}_{\boldsymbol{k}}$, $\tilde{\boldsymbol{\tau}}_{\boldsymbol{k}}$), and an upper bound on the number of commodities that can be served ($|\boldsymbol{K'}|$). For each instance, the network information includes the number of hubs ($|\boldsymbol{N}|$), the average and standard deviation of the travel time (minutes) and the distance (kilometers) between hubs ($\bar{\boldsymbol{\tau}}_{\boldsymbol{ij}}$, $\tilde{\boldsymbol{\tau}}_{\boldsymbol{ij}}$, $\bar{\boldsymbol{c}}_{\boldsymbol{ij}}$, $\tilde{\boldsymbol{c}}_{\boldsymbol{ij}}$), and the average and standard deviation of the

unloading and loading capacity of the hubs ($\bar{U}_i$, $\tilde{U}_i$, $\bar{L}_i$, $\tilde{L}_i$). Instance C and D correspond to representative days of different seasons in the same city.

Table 3.1: Demand characteristics of the test instances

Instance	Demand										
	$\|K\|$	$\|T\|$	$\bar{q}_k$	$\tilde{q}_k$	$\bar{e}_k$	$\tilde{e}_k$	$\bar{l}_k$	$\tilde{l}_k$	$\bar{\tau}_k$	$\tilde{\tau}_k$	$\|K'\|$
A	270	2.5	32	28	13:09	15	15:02	8	43	13	264
B	986	1.75	34	32	13:15	0	15:12	15	44	13	980
C	1291	5.6	29	20	13:38	55	16:58	58	60	22	1286
D	1779	5.6	29	19	13:38	55	16:56	58	56	21	

Table 3.2: Network characteristics of the test instances

Instance	Network								
	$\|N\|$	$\bar{\tau}_{ij}$	$\tilde{\tau}_{ij}$	$\bar{c}_{ij}$	$\tilde{c}_{ij}$	$\bar{U}_i$	$\tilde{U}_i$	$\bar{L}_i$	$\tilde{L}_i$
A	17	43	14	25	13	4	0	3	0
B	32	44	13	26	12	5	2	4	3
C	31	58	23	29	16	3	0	3	0
D									

3.5.2 Analysis

We compare the performance of the proposed algorithms, i.e., the IP-based heuristic (IP-H), the metaheuristic (MILS) and the hybrid matheuristic (H-MAT), on the set of instances described in Section 3.5.1. IP-H was coded in Python and uses Gurobi 8.1.1 to solve integer programs. MILS was coded in C++ and uses IBM CPLEX Optimizer 12.6 to solve integer programs. All experiments were performed in a single thread of a dedicated Intel Xeon ES-2630 2.3GHz processor with 50GB RAM running Red Hat Enterprise Linux Server 7.4.

Preliminary computational experiments were used to determine values for the parameters of MILS that ensure good quality solutions are found in under three hours for the larger instances. Based on these experiments, we set $It^O_{max} = 10, It^I_{max} = 500, T_k = -1, p = 0.05, It^I_R = 100$ and $\eta = 0.005$. For a more comprehensive analysis on how

different values of η affect the performance of MILS, see Appendix B. The maximum time allowed for solving the integer programs was set to five seconds and the incumbent solution is discarded when the time limit is exceeded and when the integrality gap is more than 10%. The maximum distance for choosing an intermediate hubs was set to $\frac{3}{4}$ of the average distance between hubs. The same set of parameter values was used when running MILS within H-MAT.

Geographic paths for commodities are allowed to use at most one intermediate hub. The integer programs solved by IP-H and H-MAT are based on a time-expanded network with a homogeneous time discretization of 2 minutes. To control the number of variables in the integer programs solved by IP-H and H-MAT, the maximum waiting time embedded in a movement arc is 15 minutes for instances A and B and 10 minutes for instances C and D. Moreover, a maximum solution time of 4 hours is imposed for each phase of the hierarchical optimization and in the second phase we seek a solution that is within 5% of optimality. MILS uses a time discretization of 1 minute.

The stages in H-MAT are defined by vector $(4, 4, 3, 3, 3)$ for instance A, $(7, 7, 6, 6, 6)$ for instance B, and $(7, 6, 6, 6, 6)$ for instances C and D. The reason for using five stages is to control the overall solution time (by invoking MILS only five times). We process fewer hubs in the later stages because finding a timed path for a commodity is more difficult towards the end and we benefit from more frequent adjustment of the geographic paths considered for commodities.

In order to assess the performance of the proposed algorithms, we report the following statistics for the best solutions produced by each algorithm: the number of commodities served ($|\boldsymbol{K^S}|$), the total travel cost (C), the number of commodities served using direct paths ($|\boldsymbol{K_d^S}|$), the number of commodities served using indirect paths ($|\boldsymbol{K_i^S}|$), the average number of segments per path for commodities served ($|\boldsymbol{G^S}|$), the average travel time (minutes) of the indirect paths for commodities served ($\boldsymbol{\tau_i^S}$), the average direct travel time for commodities served ($\boldsymbol{\tau_d^S}$), the average direct travel time for commodities not served ($\boldsymbol{\tau_d^U}$),

and the solve time, in hours ($\boldsymbol{TT}$).

Table 3.3: Performance of the Proposed Algorithms

Instance	Algorithm	$\lvert K^S \rvert$	C	$\lvert K_d^S \rvert$	$\lvert K_i^S \rvert$	$\lvert G^S \rvert$	τ_i^S	τ_d^S	τ_d^U	TT
A	MILS	264	3909.11	143	121	1.46	65.21	45.03	58.33	0.21
	IP-H	264	3900.73	149	115	1.44	62.03	42.89	60.67	0.38
	H-MAT	264	3726.94	144	120	1.45	63.58	42.96	60.17	1.56
B	MILS	965	11553.48	395	570	1.59	62.52	43.10	55.52	1.43
	IP-H	971	13339.52	542	429	1.45	63.76	41.88	65.53	15.78
	H-MAT	975	12181.09	498	477	1.49	63.18	41.35	64.36	41.50
C	MILS	1233	17408.90	606	627	1.51	79.80	58.15	67.98	2.64
	IP-H	1275	20720.57	744	531	1.42	78.22	58.10	75.38	38.08
	H-MAT	1282	18120.13	668	614	1.48	79.58	56.90	79.11	71.56
D	MILS	1626	21477.84	890	736	1.45	76.20	52.05	55.96	3.00
	IP-H	1674	20826.99	884	790	1.47	75.14	50.90	70.54	118.93
	H-MAT	1689	20059.50	851	838	1.50	74.20	51.29	69.92	174.91

The results can be found in Table 3.3. We observe that MILS is significantly faster than IP-H and H-MAT, but the efficiency comes at the price of serving fewer commodities. H-MAT produces the best solutions, but the quality comes at the price of taking a long time (175 hours for Instance D). Interestingly, H-MAT produces solutions in which the largest number of commodities is served, but also with a low cost; the cost is always less than the cost of the solutions produced by IP-H and also less than the cost of the solutions produced by MILS for Instance A and Instance D.

We also see that in the best solutions most commodities that are served are served using a direct path. The tight service constraints, i.e., the difference between due time and available time, implies that relatively few indirect paths, which involve one additional unloading operation and one additional loading operation, and, potentially, additional waiting time at the intermediate hub, are feasible. As expected, the average direct travel time of the commodities that are not served is high compared to the average direct travel time of the commodities that are served and the average travel time between hubs. Here too, this is due to the tight service constraints, as this implies that (too) few options are available for these commodities.

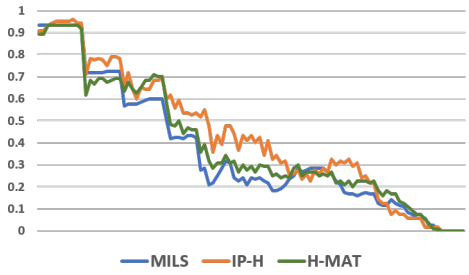
Next, we examine the differences in the solutions produced by the three algorithms

from a capacity utilization perspective. Figure 3.3 shows the average utilization of loading (or unloading) capacity across the hubs at different times during the planning period, where the capacity utilization at a point in time is computed as the number of docks occupied at that time divided by the number of docks available at that time.

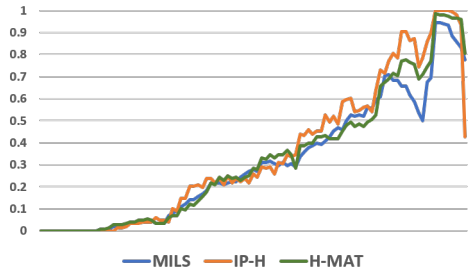
The most striking feature of the graphs shown is the difference between instances A and B and instances C and D. For Instance C and D, with a relatively long planning period, we see two loading and two unloading peaks, clearly reflecting the use of indirect timed paths, i.e., the use of hubs to transfer packages. For Instance A and B, with a relatively short planning period, we see that it is critical to get packages moving as soon as possible and the loading capacity in the early part of the planning period is almost fully utilized, and that we need all the available time as the unloading capacity in the late part of the planning period is almost fully utilized. We also observe that the solution produced by H-MAT utilizes less loading and unloading capacities than the solution produced by IP-H, while serving more commodities, which indicates that hub capacities are used more effectively.

Many interacting factors affect our ability to serve the commodities and understanding which of these factors more strongly impact our ability to serve commodities is informative and valuable. MILS is most suitable for carrying out such an investigation as it is more efficient. More specifically, we explore three scenarios, in which we relax one factor: (i) we reduce the loading and unloading time of a vehicle from 10 to 8 minutes (REDLOADUNLOADTIME), (ii) we increase the due time of each commodity by 10 minutes (INCRDUETIME), and (iii) we modify the loading and unloading capacity at each hub during the first and last ten minutes of the planning horizon by making all docks available for loading in the first ten minutes and making all docks available for unloading in the last ten minutes (ADJLOADUNLOADCAP).

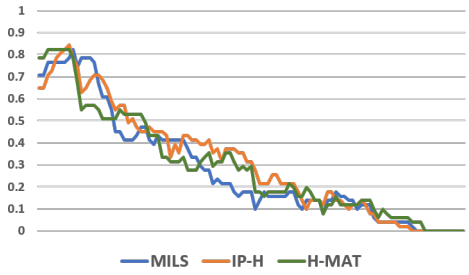
The results can be found in Table 3.4. As expected, reducing the loading and unloading times of vehicles, adjusting the loading and unloading capacities at hubs, and extending the due time of commodities all enable more commodities to be served. For Instance B, we



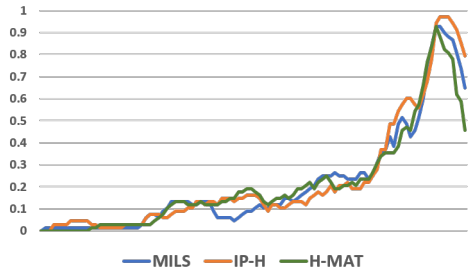
(a) Loading capacity utilization for Instance A



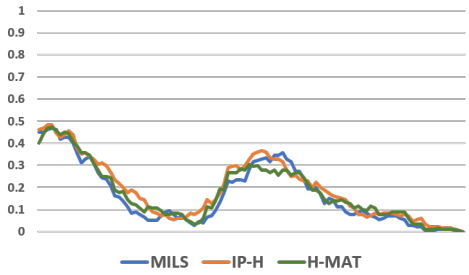
(b) Unloading capacity utilization for Instance A



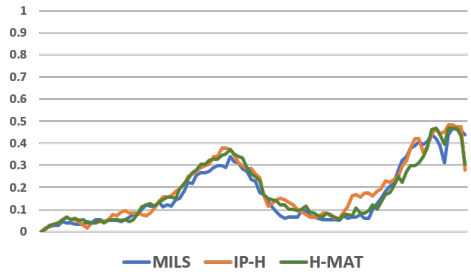
(c) Loading capacity utilization for Instance B



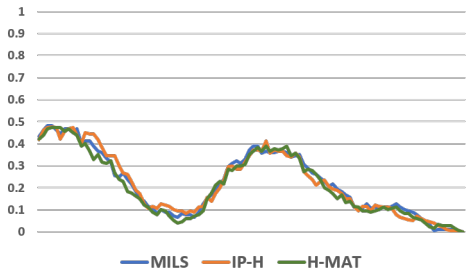
(d) Unloading capacity utilization for Instance B



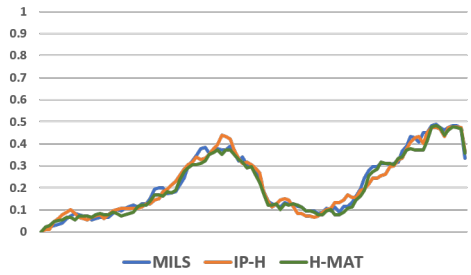
(e) Loading capacity utilization for Instance C



(f) Unloading capacity utilization for Instance C



(g) Loading capacity utilization for Instance D



(h) Unloading capacity utilization for Instance D

Figure 3.3: Average hub capacity utilization.

Table 3.4: Sensitivity Analysis for Operational Constraints

Instance	Setting	$\lvert K^S \rvert$	C	$\lvert K_d^S \rvert$	$\lvert K_i^S \rvert$	$\lvert G^S \rvert$	τ_i^S	τ_d^S	τ_d^U
B	DEFAULT	965	11553.48	395	570	1.59	62.52	43.10	55.52
	REDLOADUNLOADTIME	986	10023.49	396	590	1.60	66.69	42.78	-
	INCRDUETIME	984	9419.64	391	593	1.60	68.08	41.51	64.00
	ADJLOADUNLOADCAP	986	9998.82	400	586	1.59	66.19	42.13	-
C	DEFAULT	1233	17408.90	606	627	1.51	79.80	58.15	67.98
	REDLOADUNLOADTIME	1289	18163.38	654	635	1.49	85.33	59.02	73.50
	INCRDUETIME	1289	17999.60	649	640	1.50	84.74	58.62	61.50
	ADJLOADUNLOADCAP	1291	19677.67	705	586	1.45	84.42	59.29	-

are not only able to serve more commodities, but we are also able to do it at less cost. For Instance C, we see that minimally adjusting loading and unloading capacities at the hubs (only in the first and last ten minutes) allows the maximum number of commodities to be served, mostly because it allows many more commodities to be served using a direct path. Recall to that MILS always tries to create a direct path between pairs of hubs first, and only if a direct path is not possible due to the hub capacity constraints, it tries to create an indirect path. For this instance, we also see that serving more commodities comes at a significant cost; an increase of less than 5% in the number of commodities served comes at an increase in cost of more than 13%. Overall, these results demonstrate that serving all commodities is very challenging when hub capacity constraints are tight.

3.6 Final remarks

Same-day and instant delivery services are the fastest growing business segments of package express carriers. These services are typically offered in urban areas and require effective consolidation to be profitable. Consolidation can only be achieved by operating hub networks and because real-estate prices in urban areas are high (and are expected to go up even more), it is only possible to operate relatively small hubs and these hubs are likely to have limited loading and unloading capacity (and limited space for vehicles waiting to be loaded or unloaded). We have introduced a variant of the traditional service network design problem that incorporates loading and unloading capacity limits at hubs to study

these new delivery environments and have proposed and analyzed different heuristic solution approaches. Much more needs to done to be able to provide truly effective decision support for companies operating in this space. Most importantly, recognizing that many of the model parameters are uncertain, e.g., vehicle loading and unloading time and travel times between hubs, and developing solution approaches that go beyond using "guessing" parameter values or using point estimates as parameter values.

CHAPTER 4
INCREMENTAL NETWORK DESIGN WITH MULTI-COMMODITY FLOWS

4.1 Introduction

In incremental service network design problems, we assume that an initial service network design and a target service network design are provided, and we seek, in a fixed number of steps, to transition from the initial service network design to the target network design, where the goal is to maximize or minimize an objective function defined over the entire transition period (e.g., we want maximize the sum of the profits made in each of the transition periods). This class of problems is especially relevant for less-than-truckload and package express transportation carriers, which regularly evaluate the design and operations of their service networks prompted by a change in (forecast) demand or simply by a desire to identify cost savings. Given that a service network redesign may result in significant changes in the operations and may require significant investments, the transition to a new target design typically occurs gradually, in a number of phases. Furthermore, given that there will be a gradual transition from an initial service network to a target service network, it becomes important to properly sequence the changes as the sequence directly affects the operational costs and the profit during the entire transition period.

In this chapter, we study a novel incremental network design problem motivated by the expansion of hub capacities in package express service networks. We are given an initial and a target service network design, defined by a set of nodes, arcs, and origin-destination demands (commodities), and we seek to find a transition from the initial service network to the target service network. In the target service network design, the capacity of a subset of arcs has been increased (hub capacities can be modeled as arcs in the network). In each period, the capacity of a single arc can be increased and the cost in a period is

given by the solution to an unsplittable multi-commodity flow problem. Our objective is to find a sequence of arc capacity expansions such that the total cost during the transition is minimized.

4.1.1 Contributions

We introduce a novel incremental network design problem motivated by the expansion of hub capacities in package express service networks: the *incremental network design problem with multi-commodity flows*. We model the problem as an integer program, propose and analyze greedy heuristics and develop an exact solution approach. In the greedy heuristics, we determine the sequence of expansions by selecting arcs based on: (1) the size of expansion (static), and (2) the largest immediate reduction in flow cost (dynamic). For finding optimal solutions, we propose a depth-first search partial enumeration algorithm that explores partial sequences of expansions in a depth-first search manner, where, for each partial sequence, we compute: (1) the cost associated with the partial sequence, and (2) upper and lower bounds on the costs of the remaining transition periods, which are used to curtail the enumeration of all partial sequences. We provide worst-case analyses for the greedy heuristics and compare the efficacy of the algorithms to solving the integer programming formulation of the problem using a commercial solver. Finally, we use the proposed methodology to solve instances of the hub capacity expansion problem derived from real-world data from a large package express carrier. We also consider a variant of the problem in which temporary capacity expansions are allowed. The contributions of this research are summarized as follows:

- A new incremental network design problem is introduced, motivated by multi-phase hub capacity expansion in package express service networks;
- A number of greedy heuristics and an exact algorithm are designed, implemented, and analyzed;
- A computational study using instances derived from real-world data of a large pack-

age express carrier is conducted, and;

- A variant allowing temporary capacity expansions is analyzed.

The remainder of this chapter is organized as follows. In Section 4.2, we review relevant prior research on incremental network design and capacity expansion problems. In Section 4.3, we provide a formal description of (general) incremental network design problems and of the incremental network design problem with unsplittable multi-commodity flows. In Sections 4.4 and 4.5, we propose greedy heuristics to quickly obtain high quality solutions and an exact approach to obtain optimal solutions, respectively. In Section 4.6, we present and interpret the results of an extensive computational study. In Section 4.7, we present and discuss a variant of the incremental network design problem with unsplittable multi-commodity flows in which we are allowed to temporarily expand the capacity of arcs in the network. In Section 4.8, we conduct a computational study using a hub capacity expansion instance derived from real-world data of a large package express carrier. Finally, in Section 4.9, we finish with final remarks.

4.2 Literature Review

Since we are not aware of any literature addressing the incremental network design problem with multi-commodity flows, we briefly review literature on capacity expansion problems and on incremental network design problems.

There is a wide variety of capacity expansion problems covered in the literature for different types of service networks. For example, in the context of telecommunication networks, [63] proposes a heuristic approach for finding the least cost alternative of installing additional concentrators at the nodes and cables on the links of the network in order to meet an increasing demand. In the context of water distribution systems, [64] proposes capacity expansion alternatives to an existing water distribution system in order to improve the efficiency of water supply. For large urban transportation networks, [65] and [66] propose different strategies for opening new links and adding capacity to existing links in the network

in order to minimize congestion. In gas transportation networks, [67] presents techniques for finding the optimal location of pipeline segments to be expanded and the optimal size of these expansions such that investment costs on an existing gas transportation network are minimized. There is also a fair amount of literature purely focused on facility expansion problems concerned with the timing of facility expansions to meet increasing demand (see, e.g.,[68], [69] and [70]). These problems, however, are focused almost exclusively on finding the set of necessary expansions/changes in the service network design in order to meet the desired goals, and do not investigate how to gradually transition from the current state of the network to a provided target network design. For a more broad survey on capacity expansion problems covered in the of field Operations Research, see [71].

Incremental network design problems, which integrate both network design and scheduling problems, were originally introduced in [72] and [73]. In the context of electrical power grids, there is a fair amount of literature on incremental network design problems where the objective is to convert the current design of an electrical power grid into a smart grid, where resource and budget constraints allow only a limited number of upgrades per time period (see, e.g., [74] and [75]). Another example of incremental network design problems often studied in the literature is the one faced by managers of critical civil interdependent infrastructure systems of restoring essential public services after a non-routine event causes disruptions in the system. In this context, the objective is to determine a set of tasks to be completed, assign these tasks to work groups, and then determine the schedule of each work group to complete the tasks assigned to it such that a quality measure over the entire horizon of the restoration plan is maximized (see, e.g.,[76] and [77]).

In a setting more similar to the one we consider, [78] studies an incremental network design with maximum flows problem where, given a current network design defined over a set of nodes and arcs with fixed integer capacity, the goal is to build new arcs in the network from a set of fixed potential arcs, one at a time, such that the maximum flow in the network increases and the cumulative performance, i.e., the sum of the performance measures of the

networks in all time periods, is maximized. The authors propose different integer programming formulations for its solution and a set of greedy heuristics to find a good sequence of arcs to build in the network and meet the desired goal. The worst-case analyses for the greedy heuristics are also presented. Other incremental network design problems where, in each period, one or more arcs from a set of potential arcs are built in the network and classical network problems have to be solved with the overall goal of minimizing or maximizing performance metrics over the entire transition period include: incremental network design with shortest paths [79] and incremental network design with minimum spanning trees [80]. In these settings, the network optimization problems solved in each period are all polynomially solvable. However, the incremental network design version may be NP-complete (e.g., [79] and [81]). When a single arc is built in each period, the authors in [80] show that the incremental network design problem with minimum spanning trees can be solved efficiently using a greedy heuristic.

4.3 Problem Statement

We start by illustrating the concepts of incremental network design problems mathematically. Let x be a decision vector representing a service network design (e.g., the location and type of facilities, the shipment flow paths, etc.). Let $p \cdot x$ represent the profit of a design x and let $Ax \leq b$ represent feasibility constraints (i.e., constraints on the service network design). Finally, let x^0 and x^T represent an initial and a target network design, respectively, and let T be the number of transition periods (e.g., we want transition from an initial design to a target design in T quarters, or if a target network design has T additional hubs, we want to add the new hubs one by one). Then, we are seeking intermediate network designs x^i such that

$$x^0 \rightarrow x^1 \rightarrow x^2 \rightarrow \cdots \rightarrow x^{T-1} \rightarrow x^T.$$

Each step $x^{j-1} \rightarrow x^j$ of the transformation process corresponds to a separate design problem with constraints

$$Ax^j \leq b$$
$$|x^j - x^{j-1}| \leq \Delta,$$

where the constraint $|x^j - x^{j-1}| \leq \Delta$ reflects that only limited adjustments to design x^{j-1} can be made. Importantly, the objective covers the entire transition period, i.e.,

$$\max \sum_{i=1}^{T-1} p \cdot x^i$$

This captures the fact that we want to maximize the profit over the entire transition period, which is not necessarily the same as maximizing the profit in each individual transition period. By maximizing the increase in profit in a single period, we may put ourselves in a position that makes it difficult to achieve additional profit in the next period (because we are restricted in the changes we can make to the design in each period). Additional constraints can be added to the model representing company goals/policies, e.g., $p \cdot x^j \geq p \cdot x^{j-1}$ (profits are not allowed to decrease in any step).

Next, we provide a formal description of the incremental network design problem with multi-commodity flows. Let x^0 be the initial design representing a multi-commodity flow solution for a commodity set K to be served on a directed network $G^0 = (N, A)$ with node set N and arc set A. For each arc $a \in A$, $c_a \in \mathbb{R}_+$ represents the cost of sending one unit of flow through the arc and $u_a \in \mathbb{Z}_+$ is the (integer) capacity of the arc. Furthermore, let x^T be the target design, representing a multi-commodity flow solution for the same commodity set K on the directed network $G^T = (N, A)$ with the same set of nodes and arcs as G^0, but with a subset of arcs $\hat{A} \subseteq A$ with expanded capacity $u_a + w_a$, where $w_a \in \mathbb{Z}_+$ represents the additional capacity of arc $a \in \hat{A}$. For each $k \in K$, let $o_k \in N$ denote the commodity's origin, let $d_k \in N$ denote the commodity's destination, and let $q_k \in \mathbb{R}_+$ denote that quantity

that needs to be routed from the origin to the destination of the commodity k along a single path (i.e., we consider the unsplittable variant). We assume that $q_k \leq u_a$ for all $k \in K$. Finally, let $T = |\hat{A}|$ be the length of the transition period.

We will seek for the optimal transition of the initial network design to the target network design in T steps where, in each step, we expand the capacity of exactly one arc in $\hat{A}$ and record the multi-commodity flow costs in the modified network. The objective is to minimize the total multi-commodity flow costs over the transition period:

$$z^* = \min \sum_{t=1}^{T-1} z(x^t) \tag{4.1}$$

where $z(x^t)$ denotes the multi-commodity flow cost in the intermediate design at period t. Note that we do not consider the multi-commodity flow costs in the initial and target network designs since these are sunk costs.

The problem can be formulated as an integer program. Let $x_{a,k}^t$ be a binary decision variable representing sending all units from a commodity $k \in K$ through the arc $a \in A$ at a time period t ($x_{a,k}^t = 1$) or not ($x_{a,k}^t = 0$). Let y_a^t be another binary decision variable indicating that the arc $a \in \hat{A}$ has its capacity increased by w_a units at time t ($y_a^t = 1$) or not ($y_a^t = 0$). We formulate the incremental network design problem with multi-commodity flows as the following integer program:

$$\text{(INCMCF)} \quad \min \sum_{t=1}^{T-1} \sum_{a \in A} \sum_{k \in K} (q_k \cdot c_a) \cdot x_{a,k}^t$$

$$\sum_{a \in \delta^+(i)} x_{a,k}^t - \sum_{a \in \delta^-(i)} x_{a,k}^t = \begin{cases} 1, & i = o_k \\ -1, & i = d_k \\ 0, & \text{o.w.} \end{cases} \quad \forall i \in N, k \in K, t = 1, \ldots, T-1 \tag{4.2}$$

$$\sum_{k \in \mathcal{K}} q_k \cdot x_{a,k}^t \leq u_a, \quad \forall t = 1, \ldots, T-1,\ a \in A \setminus \hat{A} \tag{4.3}$$

$$\sum_{k \in \mathcal{K}} q_k \cdot x_{a,k}^t \leq u_a + y_a^t \cdot w_a, \quad \forall t = 1, \ldots, T-1,\ a \in \hat{A} \tag{4.4}$$

$$y_a^t \leq y_a^{t+1}, \quad \forall t = 1, \ldots, T-2,\ a \in \hat{A} \tag{4.5}$$

$$\sum_{a \in \hat{A}} y_a^t = t, \quad \forall t = 1, \ldots, T-1 \tag{4.6}$$

$$x_{a,k}^t \in \{0, 1\}, \quad \forall t = 1, \ldots, T-1,\ a \in A,\ k \in K \tag{4.7}$$

$$y_a^t \in \{0, 1\}, \quad \forall t = 1, \ldots, T-1,\ a \in \hat{A} \tag{4.8}$$

Constraints (4.2) and (4.3) are the typical integer multi-commodity flow constraints, where the flow assignment decision variables are also indexed by time period. For every arc $a \in \hat{A}$ and time period t, Constraints (4.4) ensures that the capacity of the arc is increased by w_a units if the arc is selected for expansion at or before time period t. Constraints (4.5) ensures that if the capacity of the arc $a \in \hat{A}$ is selected for expansion by time t ($y_a^t = 1$), then all decision variables $y_a^{\bar{t}}$ for $t < \bar{t} \leq T-1$ will also be set to one (i.e., $y_a^{t+1} = 1, y_a^{t+2} = 1, \ldots, y_a^{T-1} = 1$). Constraints (4.6) ensures that exactly one arc $a \in \hat{A}$ is selected for expansion at each time period. The objective function aims to minimize the total transition costs from the initial network design to the target network design, where the associated cost for each time period t is given by the solution of a integer multi-commodity flow problem. The integrality constraints of the $x_{a,t}^t$ decision variables (Constraint 4.7) ensure unsplittable flows for commodities. We denote this formulation by INCMCF.

Proposition 1. *Let $z(x^0)$ and $z(x^T)$ be the multi-commodity flow costs of the initial and target network designs, respectively, and let z^* be the optimal transition costs from the initial to the target network design (representing the $T-1$ intermediate transition periods). Then, $(T-1)z(x^T) \leq z^* \leq (T-1)z(x^0)$.*

Proof. Given that $z(x^t)$ is a monotonic non-increasing function with respect to the multi-commodity flow costs in the intermediate designs along consecutive transition periods

(which follows from $w_a \in \mathbb{Z}_+$), then the lowest and highest possible values for $z(x)$ are achieved in x^T and x^0, respectively. Thus, both sides of the inequality represent trivial lower and upper bounds for z^*. □

4.4 Greedy Heuristics

4.4.1 By size of expansion

Expanding the capacity of arcs by size of expansion, from highest to lowest, is a natural greedy strategy (GREEDYSIZEEXP-HL). A large increment in capacity suggests that more commodity paths can benefit from using the expanded capacity, which can reduce the current flow costs, but also future flow costs (i.e., in subsequent transition periods). We start by sorting the arcs in $\hat{A}$ by size of expansion, from highest to lowest. For each transition period t, we expand the capacity of a single arc in the network following the order in the sorted list and re-compute the multi-commodity flow costs in the modified network. That is, we solve $T-1$ multi-commodity flow problems, one for every transition period. The greedy heuristic is described in Algorithm 4. GREEDYSIZEEXP-HL break ties by selecting the arc with largest number of commodity paths using the arc and, if that does not resolve a tie, by selecting an arc at random (with equal probability).

Algorithm 4 GREEDYSIZEEXP-HL

input: $\hat{A}$ - set of candidate arcs for expansion
output: $\bar{c}$ - upper bound on the total transition costs

1: $\bar{c} \leftarrow 0$
2: Sort $\hat{A}$ by size of expansion, from highest to lowest
3: **for** each transition period $t = 1, \ldots, T-1$ **do**
4: $\quad \hat{a} \leftarrow \{\hat{A}_t\}$
5: $\quad u_{\hat{a}} \leftarrow u_{\hat{a}} + w_{\hat{a}}$
6: $\quad \bar{x} \leftarrow$ SolveMCF($\cdot$)
7: $\quad \bar{c} \leftarrow \bar{c} + z(\bar{x})$
8: **end for**
9: **return** $\bar{c}$

Figure 4.1a shows an example of an instance representing the worst case scenario for

GREEDYSIZEEXP-HL. In the figure, the cost of each arc is shown in black, the original capacity of each arc is shown in blue, and the size of the expansion for the three arcs to be expanded is shown in red (i.e., for arcs AB, BC and CD). There are three commodities with unit demand and origin-destination pairs: AD, BD, and CD. The initial and target network designs are depicted in Figures 4.1b and 4.1c, with flow costs $3L + 7\epsilon$ and $L + 3\epsilon$, respectively (the arcs with flow are shown in green). Increasing the capacity of arcs by size of expansion, from highest to lowest (CD, then BC, then AB), yields a total transition cost of $(3L + 5\epsilon) + (3L + 3\epsilon) = 6L + 8\epsilon$. The cheapest transition costs, however, is to expand the capacity of arcs from lowest to highest size of expansion (AB, then BC, then CD), resulting in a total cumulative flow cost of $(L + 9\epsilon) + (L + 6\epsilon) = 2L + 15\epsilon$ (we assume $L \gg 3\epsilon$). This example can be generalized to a family of worst case instances for larger values of T (and larger sets K) by adding new small rectangles of side ϵ to the base of the outer rectangle of size L as shown in in Figure 4.2. Let z_{GHL} be the total cumulative flow costs by expanding arcs by size of expansion, from highest to lowest. We can express $z(x^0)$, $z(x^T)$ and z_{GHL} in terms of T, L and ϵ, as follows:

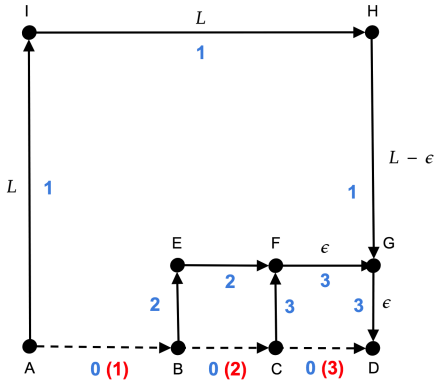
$$z(x^0) = 3L + \sum_{k=1}^{T-1}(k+2)\epsilon \tag{4.9}$$

$$z(x^T) = L + \sum_{k=1}^{T-1} k\epsilon \tag{4.10}$$

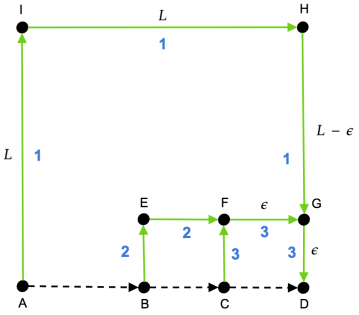
$$z_{GHL} = (T-1)\cdot(3L + \sum_{k=1}^{T-1} k\epsilon) + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon \tag{4.11}$$

Proposition 2. *In the worst case,* GREEDYSIZEEXP-HL *gets arbitrarily close to* $(T-1)z(x^0)$.

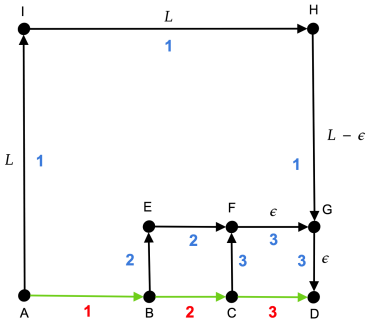
Proof. Consider the family of instances shown in Figure 4.2. By Proposition 1, we have



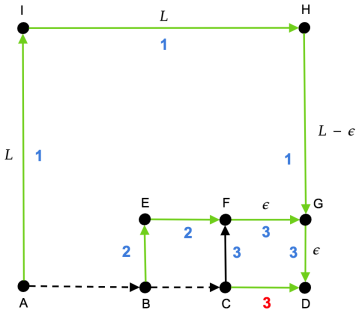
(a) Worst case example for GREEDYSIZEEXP-HL



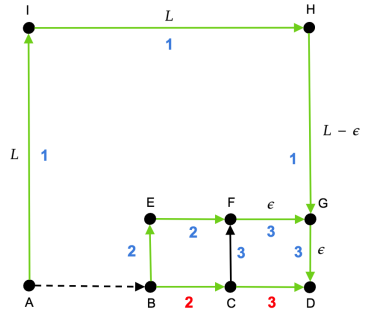
(b) Initial network design



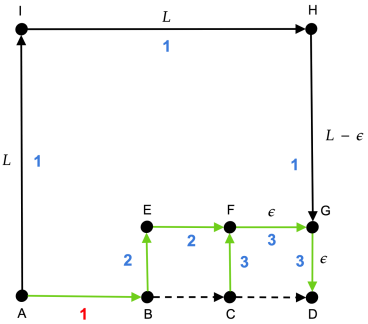
(c) Target network design



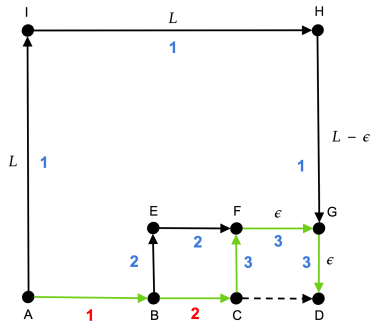
(d) First intermediate design using GREEDYSIZEEXP-HL



(e) Second intermediate design using GREEDYSIZEEXP-HL



(f) First intermediate design in the optimal solution



(g) Second intermediate design in the optimal solution

Figure 4.1: Worst case example for GREEDYSIZEEXP-HL with three origin-destination pairs with unit demand each (AD, BD and CD), showing the initial and target network designs and the intermediate designs obtained using the greedy heuristic and in the optimal solution.

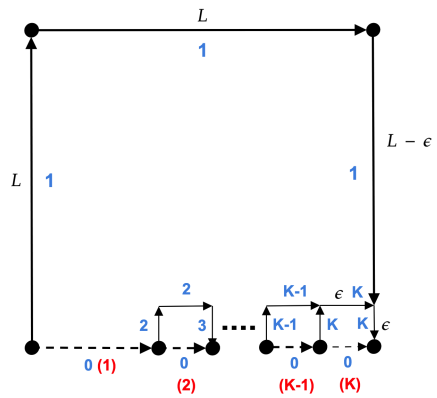


Figure 4.2: Family of worst case instances for GREEDYSIZEEXP-HL

that $z^* \leq z_{GHL} \leq (T-1)z(x^0)$. From (4.9) and (4.11) we obtain:

$$
\begin{aligned}
z_{GHL} &= (T-1)\cdot(3L + \sum_{k=1}^{T-1} k\epsilon) + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon \\
&= (T-1)\cdot(3L + \sum_{k=1}^{T-1}(k+2)\epsilon - \sum_{k=1}^{T-1} 2\epsilon) + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon \\
&= (T-1)\cdot(3L + \sum_{k=1}^{T-1}(k+2)\epsilon) - (T-1)\sum_{k=1}^{T-1} 2\epsilon + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon \\
&= (T-1)z(x^0) - \underbrace{(T-1)\sum_{k=1}^{T-1} 2\epsilon + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon}_{=\gamma\epsilon} \\
&= (T-1)z(x^0) - \gamma\epsilon \\
&\leq (T-1)z(x^0)
\end{aligned}
$$

Finally, when ϵ goes to zero:

$$
\begin{aligned}
\lim_{\epsilon\to 0} z_{GHL} &= \lim_{\epsilon\to 0} (T-1)z(x^0) - \gamma\epsilon \\
&= (T-1)z(x^0)
\end{aligned}
$$

□

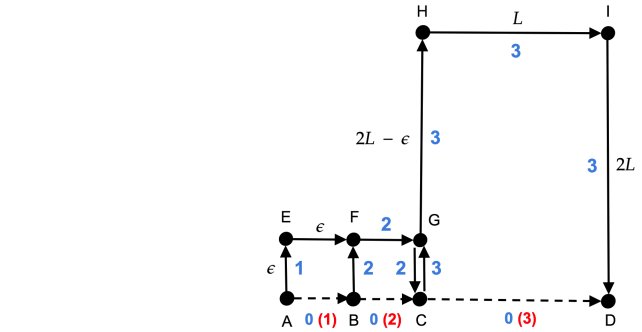
Figure 4.3 shows an example of the worst case scenario for choosing arcs for expansion

in the opposite sorting direction, from lowest to highest size of expansion, which we do in GREEDYSIZEEXP-LH. We consider again three different origin-destination pairs, with unit demand each (AD, BD and CD). Using a proof similar to the one for Proposition 2, we can show GREEDYSIZEEXP-LH also achieves a tight upper bound on the transition costs z_{GLH} of $z_{GLH} \leq (T-1)z(x^0)$ in the worst case when ϵ goes to zero.

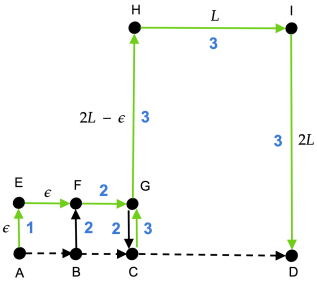
4.4.2 By largest reduction in flow costs

As we have seen, a weakness of GREEDYSIZEEXP is that there can be periods in which there is little or no reduction in the period flow costs. In order to overcome this weakness, we next explore a second greedy heuristic that focuses on the immediate reduction in flow costs rather than the size of the capacity expansion (GREEDYCOSTRED). The main idea is to identify, for each arc for which we can expand the capacity, a set of commodities that are likely to benefit from being able to use the added capacity and then choose the arc that yields the (potential) largest anticipated benefit.

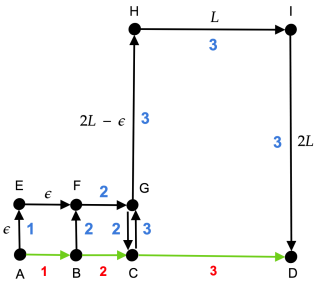
More specifically, for each transition period t, let $\hat{A}^t$ be the subset of arcs that is still available for expansion at period t, $p_{k,t-1}$ be the path assigned to a commodity $k \in K$ at period $t-1$ and $f_{a,t-1}$ be the flow going through a at period $t-1$. Furthermore, let $p^*_{k,a}$ be the shortest path connecting the origin to the destination of commodity k that uses arc a. For each candidate arc $a \in \hat{A}^t$ and commodity $k \in K$, we start by computing the difference between the cost of $p_{k,t-1}$ and the cost of $p^*_{k,a}$, i.e., $r_{k,a,t} = [c(p_{k,t-1}) - c(p^*_{k,a})]_+$. Once the values $r_{k,a,t}$ are computed for all commodities, we compute for each arc a the sum $s_{a,t} = \sum r_{k,a,t}$, starting from the highest to lowest values of $r_{k,a,t}$ until the extra capacity of w_a units has been consumed (see Algorithm 5). We then choose for expansion the arc $\hat{a} = \arg\max_{a \in \hat{A}^t} s_{a,t}$, which indicates that augmenting the capacity of $\hat{a}$ potentially yields the largest immediate reduction in flow costs. Finally, before moving on to the next transition period, we re-compute the multi-commodity flow costs in the updated network. GREEDYCOSTRED break ties by selecting arcs with the highest size of expansion first, then



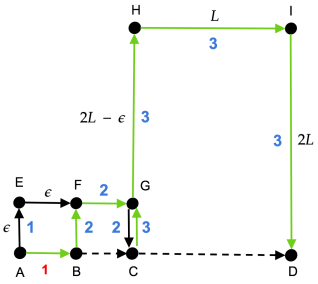
(a) Worst case example for GREEDYSIZEEXP-LH



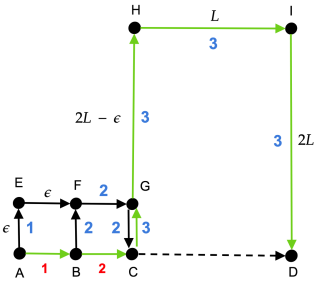
(b) Initial network design



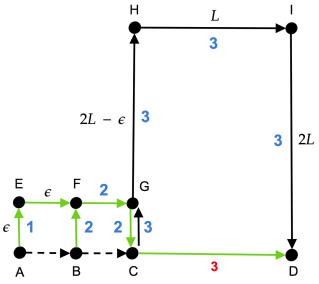
(c) Target network design



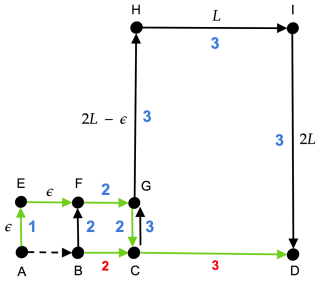
(d) First intermediate design using GREEDYSIZEEXP-LH



(e) Second intermediate design using the GREEDYSIZEEXP-LH



(f) First intermediate design in the optimal solution



(g) Second intermediate design in the optimal solution

Figure 4.3: Worst case example for GREEDYSIZEEXP-LH with three origin-destination pairs with unit demand each (AD, BD and CD), showing the initial and target network designs and the intermediate designs obtained using the greedy heuristic and in the optimal solution. Arcs used by commodity paths are highlighted in green.

by largest number of commodity paths using the arc, and finally by randomly selecting an arc. Shortest paths are pre-computed using Floyd-Warshall's algorithm in $\mathcal{O}(|N|^3)$ at the beginning of the algorithm and stored in a table for quick access.

Figures 4.4 and 4.5 illustrate the first iteration of GREEDYCOSTRED on the worst case instances for GREEDYSIZEEXP, showing that the algorithm makes the optimal selection for the first arc to expand in both cases. Figure 4.6, however, shows an example where GREEDYCOSTRED fails to find the optimal transition sequence. We consider again three different origin-destination pairs, with unit demand each (K=3, AH, BF and LM) and three candidate arcs for expansion (T=3, AB, BG and LM). The initial and target network designs are depicted in Figures 4.6b and 4.6c, with flow costs $3L + 10\epsilon$ and $L + 4\epsilon$, respectively. Increasing the capacity of arcs based on the largest immediate reduction in flow costs (LM then BG) yields a total cumulative flow cost of $6L + 10\epsilon$. The cheapest transition sequence, however, for $L > 4\epsilon$, is to expand the capacity of arcs BG then AB, resulting in a total cumulative flow cost of $4L + 16\epsilon$.

Proposition 3. *In the worst case,* GREEDYCOSTRED *gets arbitrarily close to* $(T-1)z(x^0)$.

Proof. If we generalize the instance presented in Figure 4.6 for an arbitrary number of transition periods such that we only see a large reduction in the flow costs within a multiple of L at the very last intermediate transition period, then, in each of the intermediate periods, the flow cost will decrease in multiples of ϵ. As ϵ goes to zero, the transition costs will heavily depend on L and we will see a marginal difference between the flow costs in the intermediate designs and in the initial network design, i.e., $z(x^t) \approx z(x^0)$ for all $t = 1, \ldots, T-1$. Therefore, in the worst case, GREEDYCOSTRED will get arbitrarily close to $(T-1)z(x^0)$. □

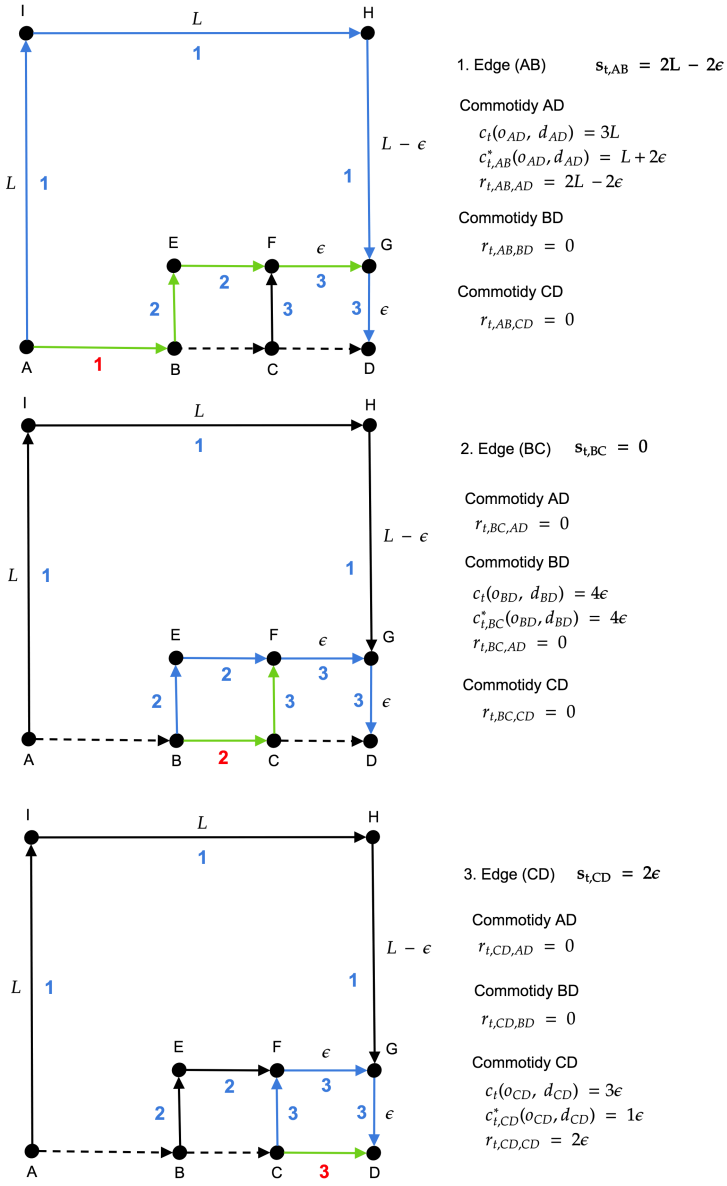


Figure 4.4: First iteration of GREEDYCOSTRED on the worst case example for GREEDYSIZEEXP-HL, showing that the algorithm chooses arc AB to expand first. Commodities that benefit from the arcs in the sequence of expansions have their original paths highlighted in blue and the shortest paths using the expanded arcs in green. arcs belonging to both paths are kept in blue.

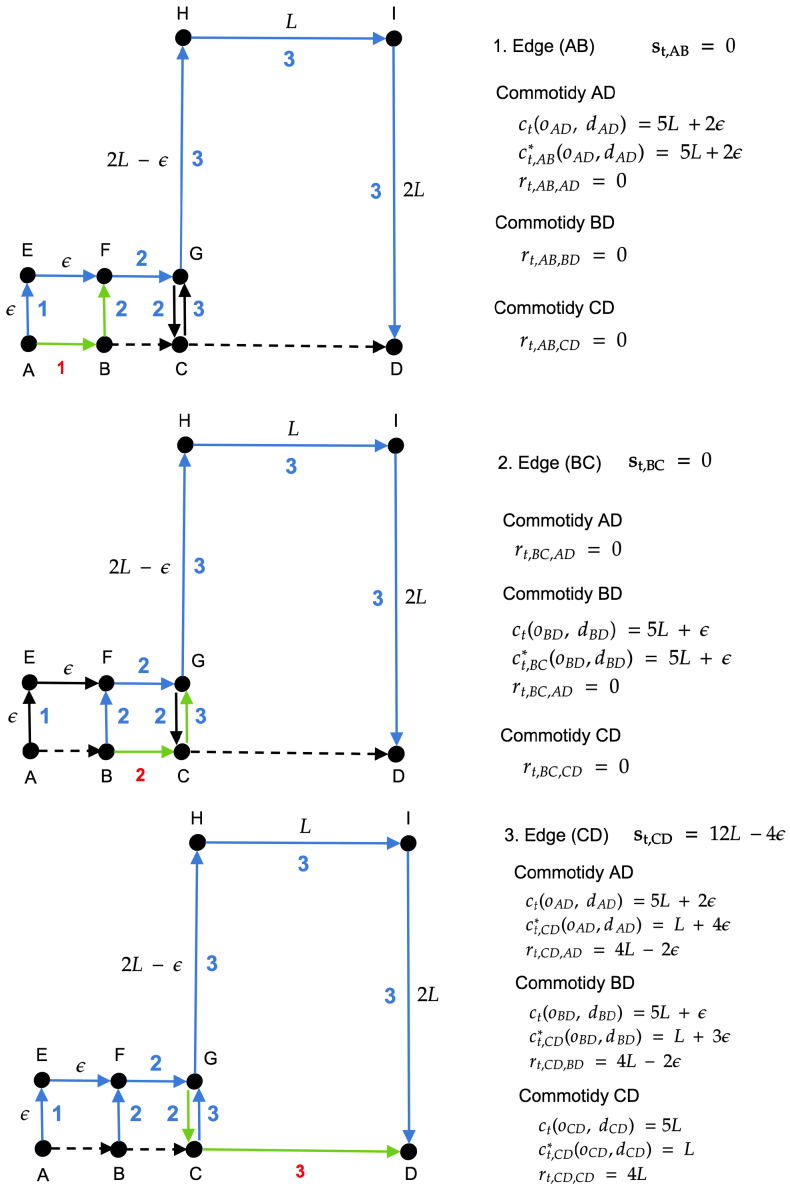
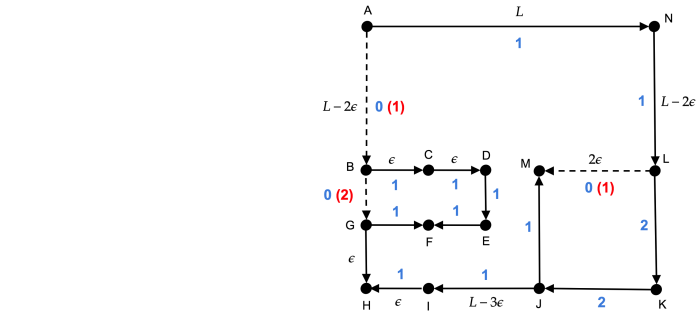
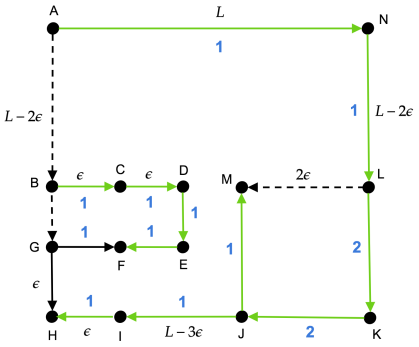


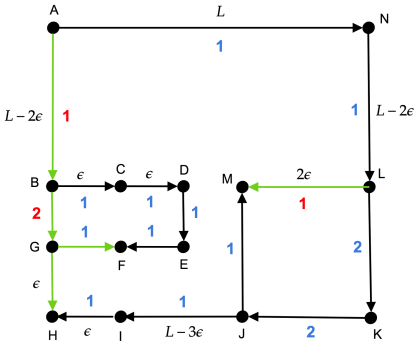
Figure 4.5: First iteration of GREEDYCOSTRED on the worst case example for GREEDYSIZEEXP-LH, showing that the algorithm chooses arc CD to expand first. Commodities that benefit from the arcs in the sequence of expansions have their original paths highlighted in blue and the shortest paths using the expanded arcs in green. arcs belonging to both paths are kept in blue.



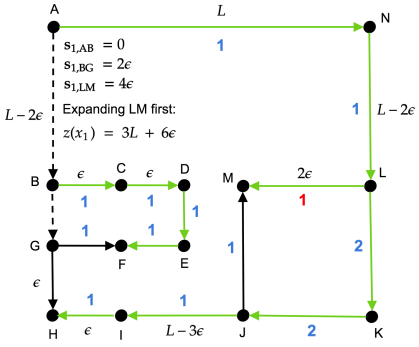
(a) Bad example for GREEDYCOSTRED



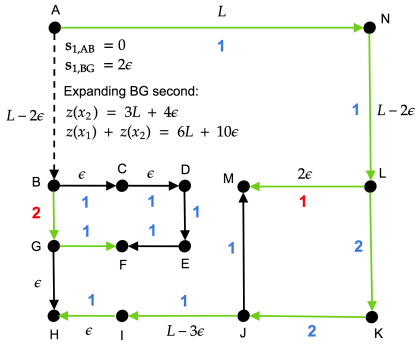
(b) Initial network design



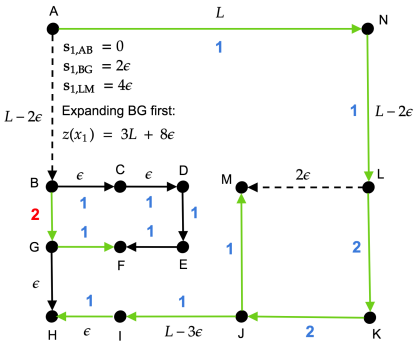
(c) Target network design



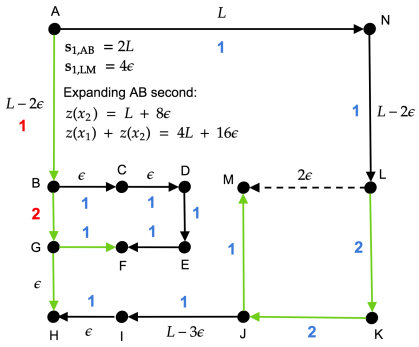
(d) First intermediate design using GREEDYCOSTRED



(e) Second intermediate design using the GREEDYCOSTRED



(f) First intermediate design in the optimal solution



(g) Second intermediate design in the optimal solution

Figure 4.6: Bad example for GREEDYSIZECOSTRED with three origin-destination pairs with unit demand each (AH, BF and LM) showing that the algorithm fails to find the optimal transition sequence. Arcs used by commodity paths are highlighted in green.

Algorithm 5 GREEDYCOSTRED

input: $\hat{A}$ - set of candidate arcs for expansion
output: $\bar{c}$ - upper bound on the total transition costs

$\bar{c} \leftarrow 0$
$\hat{A}^1 \leftarrow \hat{A}$
for each transition period $t = 1, \ldots, T-1$ **do**
  for each arc $a \in \hat{A}^t$ **do**
    for each commodity $k \in K$ **do**
      $r_{k,a,t} \leftarrow [c(p_{k,t-1}) - c(p^*_{k,a})]_+$
    end for
    $s_{a,t} \leftarrow 0$
    $\bar{u}_a \leftarrow u_a + w_a - f_{a,t-1}$
    for each commodity $k \in K$, sorted by $r_{k,a,t}$, from highest to lowest, **do**
      if $\bar{u}_a - q_k \geq 0$ **then**
        $s_{a,t} \leftarrow s_{a,t} + r_{k,a,t}$
        $\bar{u}_a \leftarrow \bar{u}_a - q_k$
      end if
    end for
  end for
  $\hat{a} \leftarrow \arg\max_{a \in \hat{A}^t} s_{a,t}$
  $u_{\hat{a}} \leftarrow u_{\hat{a}} + w_{\hat{a}}$
  $\bar{x} \leftarrow$ SolveMCF($\cdot$)
  $\bar{c} \leftarrow \bar{c} + z(\bar{x})$
  $\hat{A}^{t+1} \leftarrow \hat{A}^t \setminus \{\hat{a}\}$
end for
return $\bar{c}$

4.5 Exact method

Next, we propose a depth-first search algorithm with partial enumeration (DFSPE) for finding the optimal transition sequence of arc capacity expansions.

The algorithm enumerates transition sequences, exploring partial sequences of expansions in a depth-first search manner with a pre-specified order, where we compute, for each partial sequence: (1) the cost associated with the partial sequence, and (2) upper and lower bounds on the remaining cost of the transition sequence, used to curtail the enumeration of all possible transition sequences. The main difference between DFSPE and a standard IP-based branch and bound algorithm for solving INCMCF is that, in each iteration, DFSPE solves the unsplittable multi-commodity flow problem for a single transition period

and decides which arc to expand next following a pre-defined order, rather than branching on a single integer variable and re-solving the model considering all transition periods at once. Furthermore, given that DFSPE only considers a single intermediate design at a time, it consumes far less memory than branch and bound algorithms used in commercial solvers. This is especially true when commercial solvers keep copies of the model and of the fractional solutions at every node in the branch and bound tree. In DFSPE, the multi-commodity flow solution on an intermediate design resulting from specific subsets of $\hat{A}$ are computed on-demand and stored in tables for quick look-up. We do so in order to avoid re-computing the flow costs for intermediate designs resulting from the same subset of expanded arcs (i.e., although the transition costs may differ, the multi-commodity flow solution for an intermediate design resulting, for example, from the sequence of expansions a_1, a_2 and a_3 is the same as the one resulting from the sequence a_3, a_1, a_2. That is, the multi-commodity flow solution is the same regardless of the order in which the arcs were chosen for expansion). Hence, during the execution of the algorithm, we only keep track of the flow costs on each intermediate design and use compact and sparse data structures to store the flow information on each arc.

As in branch and bound methods, DFSPE consists of a systematic enumeration of the candidate solutions for INCMCF. The set of complete sequences of expansions are encoded in the leaf nodes of a solution tree, where the cost associated to each node at a depth $D(i)$ from the root node in the tree represents the transition costs from periods $t = 1, \ldots, D(i)$ considering the partial sequence of arcs selected for expansion in that branch. Nodes at a depth d in the tree serve as root for up to $T-d$ new branches, where each branch represents a different selection for the next arc to expand from the subset of $T - d$ remaining candidate arcs. When processing a node i at depth $D(i)$, we start, if necessary, i.e., when there is no entry in the table associated to that specific intermediate design, by computing the unsplittable multi-commodity flow costs on the intermediate design.

Before creating new branches rooted in node i, we compute lower and upper bounds on

the transition costs for the remaining subsequent periods to decide if we should continue the search down that branch or not. Let $ub(i)$ and $lb(i)$ be lower and upper bounds on the costs for the $t = D(i) + 1, \ldots, T - 1$ remaining transition periods, respectively. Also, let $c(i)$ be the transition costs of the solution associated to the partial sequence encoded in node i and ub^* be the best upper bound on the optimal solution found so far. If $c(i) + lb(i) \geq ub^*$, then we stop the search in the current branch. If, on the other hand, $c(i) + lb(i) < ub^*$, then the next step is to return to P_i, the parent node of node i, select one of the remaining arcs to expand following the pre-defined order and continue the search in a depth-first manner. In order to do so, we choose the arc that potentially results in the largest immediate reduction in the multi-commodity flow costs, using the same procedure described in Section 4.4.2. The algorithm break ties by choosing the arc that yields the lowest upper bound on the remaining transition costs.

Whenever there are no more branches to explore in a given node, the algorithm then resumes the search from the parent node following the pre-specified order of arcs considered for expansion. The pseudo-code for the recursive implementation of DFSPE is shown in Algorithm 6. The heuristic procedures for obtaining upper and lower bounds are described next.

4.5.1 Computing upper and lower bounds

The effectiveness of DFSPE depends on the efficient determination of lower and upper bounds, both in terms of the quality and speed, which can lead to an exhaustive search in the solution tree when no branches of the tree are pruned. The challenge of obtaining fast high quality bounds is twofold: (1) choosing a good sequence of the remaining candidate arcs to expand, and (2) obtaining a fast and good approximation on the optimal solution for the unsplittable multi-commodity flow problem in the intermediate designs along the remaining transition periods. Next, we present fast heuristic approaches for obtaining such bounds.

Algorithm 6 Depth-first search algorithm with partial enumeration (DFSPE)

input: i - node index
$\hat{A}^i$ - set of remaining candidate arcs for expansion
$c(P_i)$ - solution cost at the parent node

if $|\hat{A}^i| = 1$ **then**
  if $c(P_i) < ub^*$ **then**
    $ub^* \leftarrow c(P_i)$
  end if
  return
else
  for each arc $a \in \hat{A}^i$ sorted by potential immediate reduction in flow costs, from highest to lowest **do**
    $u_a \leftarrow u_a + w_a$
    $\bar{x} \leftarrow$ SolveMCF($\cdot$)
    $c(i) \leftarrow c(P_i) + z(\bar{x})$
    $ub(i) \leftarrow$ UB-H$(z(\bar{x}), \hat{A}^i \setminus \{a\})$
    $lb(i) \leftarrow$ LB-H$(\hat{A}^i \setminus \{a\})$
    if $c(i) + ub(i) < ub^*$ **then**
      $ub^* \leftarrow c(i) + ub(i)$
    end if
    if $c(i) + lb(i) < ub^*$ **then**
      $j \leftarrow$ create child node
      DFSPE$(j, \hat{A}^i \setminus \{a\}, c(i))$
    end if
    $u_a \leftarrow u_a - w_a$
  end for
  return
end if

Upper bounds

By assuming a specific remaining transition sequence of arcs selected for expansion, and by heuristically solving the multi-commodity flow problem in each of the intermediate designs, we obtain an upper bound on the cost of the remaining transition periods.

For each of the remaining transition periods $t = D(i)+1, \ldots, T-1$ in a node i, we will use the same policy for deciding the next arc to expand next, i.e., by the potential largest immediate reduction in flow costs. Once an arc is selected for expansion, the next step is to obtain an upper bound on the multi-commodity flow costs in the resulting intermediate design.

For each commodity $k \in K$ and period t, let us consider again $p_{k,t-1}$ as the path assigned to k at period $t-1$, $p^*_{k,a}$ as the shortest path connecting the origin to the destination of k going through the arc a, and $f_{a,t-1}$ as the flow going through a at period $t-1$, where the arc a represents the arc selected for expansion at period t. Let us also consider p^*_k as the shortest path connecting the origin to the destination of k. The heuristic is divided in two parts: Phase 1 and Phase 2. In Phase 1, we first compute $r_{k,a,t} = [c(p_{k,t-1}) - c(p^*_{k,a})]_+$ for all commodities $k \in K$. We then start pushing commodities through $p^*_{k,a}$, starting from the highest to lowest values of $r_{k,a,t}$, whenever possible, until the expanded arc becomes saturated. When doing so, we always update the residual capacities of the arcs throughout the network. That is, a flow of size q_k is removed from all arcs in $p_{k,t-1}$ and added to all arcs in $p^*_{k,a}$.

Let $\bar{K}$ be the subset of K that contains the commodities that, either: (1) could not fit in one or more arcs in $p^*_{k,a}$, or (2) satisfy $c(p_{k,t-1}) < c(p^*_{k,a})$. Note that when we successfully push commodities in cheaper paths using arc a in Phase 1, new and cheaper paths may become available for the commodities in $\bar{K}$. Therefore, in Phase 2, for each commodity $k \in \bar{K}$ sorted by the highest to lowest values of $r_{k,t,*} = [c(p_{k,t-1}) - c(p^*_k)]_+$, we try to push the commodity in the shortest path $p^*_{k,t}$ connecting the origin to the destination of k where we only consider arcs with enough residual capacity left to accommodate k at period t, always updating the residual capacities of the arcs throughout the network and recording the difference $r_{k,t} = [c(p_{k,t-1}) - c(p^*_{k,t})]_+$.

Let $s_{a,t}$ and s_t, be, respectively, the sum of the values of $r_{a,k,t}$ and $r_{k,t}$ only considering the commodities that were successfully pushed in the cheaper paths in Phase 1 and Phase 2. Therefore, an upper bound on the optimal solution cost for the multi-commodity flow problem on the intermediate design at period t is given by:

$$\bar{z}(x^t) = \bar{z}(x^{t-1}) - s_{a,t} - s_t$$

We repeat Phase 1 and Phase 2 until all transition periods have been processed, where, at the beginning of each period, a single arc $a \in \hat{A}$ is selected for expansion. The upper bound on the remaining transition costs at a node i is then defined as:

$$ub(i) = \sum_{t=D(i)}^{T-1} \bar{z}(x^t)$$

The step-by-step of this procedure is outlined in Algorithm 7. Although the shortest paths $p^*_{k,a}$ and p^*_k can be pre-computed, in the worst case one additional shortest path is computed for every commodity k and period t using Bellman-Ford's algorithm in $\mathcal{O}(|N| \cdot |A|)$ steps. Therefore, the algorithm runs in $\mathcal{O}(T \cdot |K| \cdot |N| \cdot |A|)$ steps. Observe that UB-H is also a greedy heuristic; arcs are selected for expansion based on a greedy policy and commodities are pushed in new paths starting from the ones that could benefit the most from being re-routed in the network. The algorithm assures that the solution obtained for the multi-commodity flow problem in each transition period is feasible, given that the commodity paths either remain the same or are replaced by cheaper paths, always respecting the arc capacity constraints. In the worst case, when both the sequence of expansions and the order of commodities processed in each period are poorly chosen, then the total transition costs will get arbitrarily close to the worst case bound of $(T-1)z(x^0)$.

Lower bounds

We obtain a lower bound, by (1) assuming a best-case scenario, i.e., in each of the remaining transition periods the multi-commodity flow cost of the target design is achieved, and (2) computing the linear relaxation of INCMCF only considering the remaining transition periods.

In the linear relaxation of INCMCF, we relax the integrality constraints of the decision variables $x^t_{e,k}$ and y^t_e and only consider the $T - D(i) - 2$ remaining transition periods. The initial network design is set to be the current network design associated to the node i and

Algorithm 7 Upper bound heuristic (UB-H)

input: $\bar{c}$ - flow costs on the current intermediate design
$\hat{A}$ - set of remaining candidate arcs for expansion
output: ub - upper bound on the remaining transition costs

$ub \leftarrow 0$
$z \leftarrow \bar{c}$
$t \leftarrow 1$
while $|\hat{A}| > 1$ **do**
  $a \leftarrow$ choose arc for expansion based on the largest immediate flow cost reduction
  $\hat{A} \leftarrow \hat{A} \setminus \{a\}$
  $u_a \leftarrow u_a + w_a$
  for each commodity $k \in K$ **do** # Phase 1
    $r_{k,a,t} \leftarrow [c(p_{k,t-1}) - c(p^*_{k,a})]_+$
    $r_{k,t,*} \leftarrow [c(p_{k,t-1}) - c(p^*_k)]_+$
  end for
  $\bar{K} \leftarrow \{\}$
  $s_{a,t} \leftarrow 0$
  for each commodity $k \in K$, sorted by $r_{k,a,t}$, from highest to lowest, **do**
    if k fits in all arcs of $p^*_{k,a}$ **then**
      push k in $p^*_{k,a}$ and update residual capacities of the arcs in $p_{k,t}$
      $s_{a,t} \leftarrow s_{a,t} + r_{k,a,t}$
    else
      $\bar{K} \leftarrow \bar{K} \cup \{k\}$
    end if
  end for
  $s_t \leftarrow 0$
  for each commodity $k \in \bar{K}$, sorted by $r_{k,t,*}$ from highest to lowest **do** # Phase 2
    if k fits in all arcs of $p^*_{k,t}$ **then**
      $r_{k,t} \leftarrow [c(p_{k,t-1}) - c(p^*_{k,t})]_+$
      push k in $p^*_{k,t}$ and update residual capacities of the arcs in $p_{k,t-1}$
      $s_t \leftarrow s_t + r_{k,t}$
    end if
  end for
  $z \leftarrow z - s_{a,t} - s_t$
  $ub \leftarrow ub + z$
  $t \leftarrow t + 1$
end while
return ub

the set of candidate arcs of expansion is limited to the arcs not yet selected for expansion by period $t = D(i)$. That is, we solve smaller linear programs adjusting the variables and constraints accordingly.

Let $z(x^{[D(i)+1,T-1]})^{LP}$ be the optimal solution cost of the linear relaxation of INCMCF for the remaining transition periods, starting from the current intermediate design $x^{D(i)}$. From Proposition 1, we saw that a valid lower bound on the transition costs for the $(T-1)$ transition periods is $(T-1) \cdot z(x^T)$. Therefore, given that the integrality gap of the linear relaxation of INCMCF may be large, we set the lower bound $lb(i)$ as:

$$lb(i) = \max\{z(x^{[D(i)+1,T-1]})^{LP}, (T - D(i) - 2) \cdot z(x^T)\}$$

that is, we take the maximum between the lower bound obtained by solving the linear relaxation of INCMCF and the transition costs assuming the best case scenario where we obtain the cheapest possible flow costs in all subsequent periods.

4.6 Computational Study

4.6.1 Instances

The proposed algorithms were used to solve a set of randomly generated instances representing different initial and target network designs.

For a given number of nodes $n = |N|$, we start by randomly sampling the x and y coordinates of the nodes in N on a grid of size $4n \times 4n$. Arcs between nodes are only created if the euclidean distance between nodes is within a given threshold of the largest euclidean distance between any pair of nodes. The capacity of an arc is chosen uniform randomly from $[Q, 2Q]$. The number of commodities is chosen uniform randomly from $[\frac{n(n-1)}{2}, n(n-1)]$. The origin and the destination nodes of each commodity k are chosen randomly and the quantity q_k associated with the commodity is chosen uniform randomly from $[1, Q]$.

For each instance, we obtain the target network design by solving the integer programming model below, where $x_{a,k}$ and y_a are integer decision variables representing, respectively, sending commodity k through arc a ($x_{a,k} = 1$) or not ($x_{a,k} = 0$) and the size of the capacity expansion of arc a:

$$\text{(TND)} \quad \min \sum_{a \in A} \bar{c}_a \cdot y_a$$

$$s.t. \quad \sum_{a \in \delta^+(i)} x_{a,k} - \sum_{a \in \delta^-(i)} x_{a,k} = \begin{cases} 1, i = o_k \\ -1, i = d_k \\ 0, \text{o.w} \end{cases} \quad \forall i \in N, k \in K \tag{4.12}$$

$$\sum_{k \in \mathcal{K}} q_k \cdot x_{a,k} \leq u_a + y_a, \forall a \in A \tag{4.13}$$

$$\sum_{a \in A} \sum_{k \in K} (q_k \cdot c_a) \cdot x_{a,k} \leq \alpha \cdot z(x^0) \tag{4.14}$$

$$y_a \leq 2Q \cdot w_a, \forall\ a \in A \tag{4.15}$$

$$\sum_{a \in A} z_a \leq n \tag{4.16}$$

$$x_{a,k} \in \{0,1\}, \forall\ a \in A, k \in K \tag{4.17}$$

$$y_a \in \mathbb{Z}_+, \forall\ a \in A \tag{4.18}$$

$$z_a \in \{0,1\}, \forall\ a \in A \tag{4.19}$$

The per-unit cost of using an arc, c_a, is set to be a fraction of its length (the euclidean distance between its tail and its head) and the per-unit cost of expanding the capacity of an arc, $\bar{c}_a$, is set to ρc_a with $\rho > 1$. Constraints (4.12) and (4.13) are the typical multi-commodity flow constraints, where now the capacity of each arc a can be expanded by y_a units. Constraint (4.14) restricts that the multi-commodity flow costs in the target design resulting from the arc capacity expansions must be at most a fraction $\alpha \in (0,1)$ from the flow costs in the initial network design. Finally, constraint (4.15) limits the maximum

increase in capacity for each arc to $2Q$ units and constraint (4.16) limits the number of arcs that can have increased capacities.

Table 4.1 summarizes the characteristics of the thirteen instances generated. We report: the number of nodes ($|\boldsymbol{N}|$), the number of arcs ($|\boldsymbol{A}|$), the number of expanded arcs in the target design ($|\hat{\boldsymbol{A}}|$), the number of commodities ($|\boldsymbol{K}|$), the average direct distance between connected nodes ($\boldsymbol{Avg_D}$), the longest direct distance between connected nodes ($\boldsymbol{L_D}$), the cost of the initial design ($\boldsymbol{z(x^0)}$), the cost of the target design ($\boldsymbol{z(x^T)}$) and the time it takes to solve the unsplittable multi-commodity flow problem to obtain the initial design, in seconds ($\boldsymbol{TT_{z(x^0)}}$).

Table 4.1: Instances generated for the incremental network design problem with multi-commodity flows

Instance	$\lvert N\rvert$	$\lvert A\rvert$	$\lvert\hat{A}\rvert$	$\lvert K\rvert$	Avg_D	L_D	$z(x^0)$	$z(x^T)$	$TT_{z(x^0)}$
S1	8	57	6	44	5.93	10.05	68.17	61.35	0.07
S2	10	65	9	51	8.09	13.00	165.80	155.64	0.14
S3	12	125	6	79	11.12	19.31	290.21	261.13	0.38
S4	12	99	17	124	10.45	17.26	468.35	448.38	0.72
M1	16	219	10	183	12.61	23.02	769.11	692.19	3.12
M2	16	223	22	227	15.77	28.32	1,027.26	924.47	4.12
M3	16	233	18	203	17.02	28.32	930.68	837.25	5.75
M4	16	229	22	216	18.38	32.02	1,271.06	1,143.96	5.14
L1	16	183	35	208	12.48	20.62	999.05	948.97	8.37
L2	16	209	35	229	15.26	29.21	1,201.41	1,087.00	2.97
L3	20	341	17	370	14.92	27.78	1,736.65	1,566.17	109.41
L4	20	283	55	321	16.82	28.46	2,133.74	2,035.60	179.27
L5	32	893	58	384	33.21	57.69	7,507.12	7,132.49	120.42

4.6.2 Analysis

The goal of our computational study is twofold. First, we want to compare the performance of the proposed greedy heuristics, namely GREEDYSIZEEXP-HL, GREEDYSIZEEXP-LH, and GREEDYCOSTRED. Second, we want to compare the performance of DFSPE and solving the integer programming formulation using a commercial solver. The algorithms

were coded in C++ and use IBM CPLEX Optimizer 12.7 to solve integer programs. All experiments were conducted in a single thread of a dedicated Intel Xeon ES-2630 2.3GHz with 250GB RAM, running Red Hat Enterprise Linux Server 7.6.

Table 4.2 reports the following statistics for the solutions produced by the greedy heuristics: the cost of the first intermediate design ($z(x^1)$), the cost of the last intermediate design ($z(x^{T-1})$, the average cost per period ($\overline{z(x)}$), the average running time per period ($\overline{TT}$), and the total running time of the algorithm (TT). We observe that GREEDYSIZEEXP-HL outperforms GREEDYSIZEEXP-LH for all instances, which confirms our intuition that selecting arcs for expansion based on size of expansion, from highest to lowest, produces better solutions than selecting arcs in the opposite order. However, we observe too that choosing arcs for expansion based on the potential immediate flow cost reduction, as in GREEDYCOSTRED, results in even better solutions (with the best performance in ten out of the thirteen instances). We note that the flow costs in the first intermediate designs in the best solutions found are all equal or lower than the ones in the solutions with higher cost. It is also interesting to observe how the sequence of arc capacity expansions affects the running time of the algorithms. In instances M4, L3 and L4, for example, the time required for evaluating the complete sequence of expansions determined by GREEDYSIZEEXP-LH takes, on average, 2.65 times longer than evaluating the sequence of expansions determined by GREEDYCOSTRED. This is due to differences in the unsplittable multi-commodity flow problems that need to be solved for different sequences of expansions.

Next, we assess the performance of the exact methods. In Table 4.3, we report following statistics for the solutions produced by CPLEX: the average cost per period of the solution to the linear relaxation of INCMCF ($\overline{z(x)}^{LP}$), the integrality gap (i.e., ${}^{z(x)^{OPT} - z(x)^{LP}}/_{z(x)^{LP}}$), as a percentage ($G_{INT}(\%)$), the number of nodes explored in the search tree ($\#Nodes$), the number of integer solutions found ($\#Sols$), the average cost per period of the first integer solution found ($\overline{z(x)}^{1}$), the average cost per period over all integer solutions found ($Avg\ \overline{z(x)}$), the average cost per period of the best integer solution found ($\overline{z(x)}^{B}$), the

Table 4.2: Computational results for the greedy heuristics. The best solutions are highlighted in bold.

Instance	Algorithm	$z(x^1)$	$z(x^{T-1})$	$\overline{z(x)}$	$\overline{TT}$	TT
S1	GREEDYSIZEEXP-LH	68.17	64.22	66.67	0.05	0.25
	GREEDYSIZEEXP-HL	66.68	61.55	64.21	0.05	0.26
	GREEDYCOSTRED	66.68	61.55	**63.34**	0.05	0.27
S2	GREEDYSIZEEXP-LH	165.55	158.47	162.03	0.07	0.57
	GREEDYSIZEEXP-HL	161.35	155.89	**158.87**	0.07	0.52
	GREEDYCOSTRED	161.35	155.89	**158.87**	0.07	0.54
S3	GREEDYSIZEEXP-LH	284.23	271.85	279.43	0.22	1.09
	GREEDYSIZEEXP-HL	278.51	265.01	**272.27**	0.20	0.99
	GREEDYCOSTRED	278.51	265.01	**272.27**	0.21	1.05
S4	GREEDYSIZEEXP-LH	468.35	451.72	462.50	0.39	6.55
	GREEDYSIZEEXP-HL	466.21	448.98	455.84	0.52	8.77
	GREEDYCOSTRED	466.21	448.98	**455.63**	0.47	8.03
M1	GREEDYSIZEEXP-LH	766.21	708.40	742.43	2.03	18.31
	GREEDYSIZEEXP-HL	751.23	695.02	717.33	1.55	13.94
	GREEDYCOSTRED	751.23	695.02	**717.23**	1.90	17.08
M2	GREEDYSIZEEXP-LH	1,027.26	937.63	986.35	3.26	68.41
	GREEDYSIZEEXP-HL	1,015.37	924.90	967.88	3.47	72.77
	GREEDYCOSTRED	1,022.84	926.61	**976.00**	3.65	76.71
M3	GREEDYSIZEEXP-LH	930.67	856.61	906.55	2.83	48.04
	GREEDYSIZEEXP-HL	913.53	839.27	868.24	2.30	39.09
	GREEDYCOSTRED	913.53	841.25	**865.98**	1.82	30.99
M4	GREEDYSIZEEXP-LH	1,271.06	1,189.10	1,250.69	3.81	80.11
	GREEDYSIZEEXP-HL	1,227.00	1,144.54	**1,161.17**	1.70	35.6
	GREEDYCOSTRED	1,227.00	1,144.54	1,165.11	1.95	40.97
L1	GREEDYSIZEEXP-LH	999.06	956.26	983.75	3.88	131.81
	GREEDYSIZEEXP-HL	993.91	949.52	964.46	3.19	108.46
	GREEDYCOSTRED	993.91	950.90	**963.11**	3.17	107.69
L2	GREEDYSIZEEXP-LH	1,200.64	1,090.16	1,156.62	2.28	77.5
	GREEDYSIZEEXP-HL	1,196.83	1,088.05	1,119.25	2.32	78.83
	GREEDYCOSTRED	1,192.39	1,096.41	**1,115.92**	2.06	70.01
L3	GREEDYSIZEEXP-LH	1,731.99	1,597.81	1,685.09	18.65	298.45
	GREEDYSIZEEXP-HL	1,696.38	1,571.54	1,604.68	7.82	125.16
	GREEDYCOSTRED	1,696.38	1,567.74	**1,601.57**	7.07	113.08
L4	GREEDYSIZEEXP-LH	2,133.74	2,037.12	2,101.08	39.50	2,133.10
	GREEDYSIZEEXP-HL	2,129.30	2,036.63	2,065.49	22.36	1,207.39
	GREEDYCOSTRED	2,126.44	2,036.27	**2,059.09**	11.47	619.22
L5	GREEDYSIZEEXP-LH	7,507.12	7,188.57	7,417.02	16.94	965.37
	GREEDYSIZEEXP-HL	7,455.29	7,143.49	**7,225.52**	9.70	552.88
	GREEDYCOSTRED	7,455.29	7,143.49	7,232.27	10.68	608.88

optimality gap between the best bound $z(x)^{BB}$ and the best integer solution found as reported by CPLEX (i.e., $z(x)^{B} - z(x)^{BB}/z(x)^{BB}$), as a percentage ($\boldsymbol{G_{OPT}(\%)}$), and the total time of the algorithm ($\boldsymbol{TT}$), where we set a time limit of 24 hours (86,400 seconds).

Table 4.3: Computational results when solving the integer programming formulation using CPLEX.

Instance	$\overline{z(x)}^{LP}$	$G_{INT}(\%)$	$\#Nodes$	$\#Sols$	$\overline{z(x)}^{1}$	$Avg\,\overline{z(x)}$	$\overline{z(x)}^{B}$	$G_{OPT}(\%)$	TT
S1	61.92	2.29	3	1	63.34	63.34	63.34	0.00	0.46
S2	156.57	0.90	3	1	157.98	157.98	157.98	0.00	0.86
S3	250.58	7.50	1	1	269.39	269.39	269.39	0.00	0.94
S4	423.09	0.95	230	5	427.26	427.18	427.13	0.00	63.8
M1	690.84	3.48	1,412	8	718.91	716.73	715.78	0.00	179.53
M2	923.49	2.96	322,791	18	952.13	951.87	951.74	0.00	82,017.62
M3	835.26	3.21	4,282	7	880.20	867.09	862.11	0.00	979.96
M4	1,143.98	1.35	857	4	1,161.28	1,160.32	1,159.43	0.00	245.17
L1	952.02	-	113,963	0	-	-	-	-	86,400.00
L2	959.39	-	168,321	0	-	-	-	-	86,400.00
L3	1,566.01	2.13	31,678	9	1,599.96	1,599.63	1,599.50	0.00	13,616.60
L4	2,054.56	-	17,162	0	-	-	-	-	86,400.00
L5	7,151.46	-	51,719	1	7,316.06	7,316.06	7,316.06	1.77	86,400.00

We observe that CPLEX was not able to find the optimal solutions for four out of the five large instances (L1, L2, L4 and L5) within the time limit of 24 hours. In fact, no integer feasible solutions were found for three out of these four instances (L1, L2 and L4), and a single integer solution was found for instance L5. We also observe that the integrality gaps (when available) are generally small, 2.70%, on average. For the instances where more than one integer solution was found, we see that the difference between the cost of the first solution found and the cost of the last solution found is very small, 0.4% on average, which is true in part because of the small integrality gaps. We also observe that the relationship between the running times and the size of the instances in terms of the number expanded arcs is not that clear. For example, instance M2 has roughly the same number of nodes, arcs, expanded arcs and commodities as instance M4, but CPLEX takes roughly $334\times$ longer to find its optimal solution than for M4. Instance L3, on the other hand, is larger than M2 in all aspects, but CPLEX finds its optimal solution roughly 6 times faster.

The results for DFSPE are summarized in Table 4.4. We report: the percentage of nodes (partial sequences) explored, out of the $2^{|\hat{A}|} - 2$ nodes in the tree ($\boldsymbol{\#PS(\%)}$), the number of complete sequences explored, i.e., the number of solutions found ($\boldsymbol{\#Sols}$), the

percentage of time spent solving LPs ($\boldsymbol{TT_{LP}(\%)}$), the percentage of time spent computing upper bounds ($\boldsymbol{TT_{UB}(\%)}$), the percentage of time spent solving IPs ($\boldsymbol{TT_{IP}(\%)}$), the average cost per period of the first complete sequence explored ($\overline{\boldsymbol{z(x)}}^{\boldsymbol{1}}$), the average cost per period over all complete sequences explored ($\boldsymbol{Avg\ \overline{z(x)}}$), the average cost per period of the best solution found ($\overline{\boldsymbol{z(x)}}^{\boldsymbol{B}}$), and the total running time of the algorithm ($\boldsymbol{TT}$), where we again set a time limit of 24 hours (86,400 seconds).

Table 4.4: Computational results for DFSPE.

Instance	$\#PS(\%)$	$\#Sols$	$TT_{LB}(\%)$	$TT_{UB}(\%)$	$TT_{IP}(\%)$	$\overline{z(x)}^{1}$	$Avg\ \overline{z(x)}$	$\overline{z(x)}^{B}$	TT
S1	72.59	1	13.61	1.48	84.91	63.34	63.34	63.34	1.34
S2	36.67	93	25.59	1.78	72.63	158.58	158.14	157.98	16.59
S3	80.64	29	16.71	1.26	82.04	272.27	270.61	269.39	11.3
S4	28.21	1219	40.37	0.77	58.86	428.84	427.53	427.13	817.6
M1	71.03	179	25.10	2.92	71.98	717.23	716.11	715.78	1,309.14
M2	0.97	6,663	39.12	5.54	55.34	974.85	968.10	951.74	86,400.00
M3	5.00	5,973	37.67	8.96	53.37	865.98	864.03	862.11	86,400.00
M4	0.67	1,265	46.99	8.29	44.72	1,165.08	1,162.21	1,160.82	86,400.00
L1	<0.01	3,277	38.05	5.60	56.34	962.86	962.36	962.20	86,400.00
L2	<0.01	1,671	59.39	2.63	37.97	1,115.27	1,110.96	1,110.16	86,400.00
L3	6.77	91	39.08	1.26	59.66	1,601.58	1,600.38	1,599.50	86,400.00
L4	<0.01	717	64.23	2.21	33.56	2,059.09	2,058.87	2,058.70	86,400.00
L5	<0.01	591	56.04	7.39	36.58	7,231.98	7,230.75	7,223.08	86,400.00

The first thing to notice is that the algorithm takes considerably more time for finding the optimal solutions for the small and medium size instances than CPLEX. Recall that DFSPE explores partial sequences of expansions, where, in each iteration, the algorithm selects a single arc for expansion based on a pre-defined order and computes the unsplittable multi-commodity flow problem in the current intermediate design, as well as lower and upper bounds on the remaining transition costs. Therefore, the first complete sequence of expansions is only explored after solving $T - 1$ integer and linear programs, which directly affects the running times especially for the small instances. Looking at the total percentage of partial sequences explored for the instances where an optimal solution was found in under 24 hours, we see that the lower and upper bounds computed at each node manage to eliminate big portions of the solution tree. For instances S2 and S4, for example, around 63% and 72% of the nodes in the solution tree were pruned, respectively. We also observe that the algorithm takes a considerable amount of time solving linear programs

when computing lower bounds (38% on average, with up to 64.23% for instance L4). On the other hand, the percentage of time spent computing upper bounds is small (less than 10%). Even though the algorithm finds a large number of feasible solutions (especially for the large instances), we observe that, for all instances, the gaps between the costs of the first solutions found and of the average solutions are small (0.23%, on average). This suggests that there are likely many solutions (unsplittable flows) that either have the same cost or have only a small cost difference.

Next, we analyse the quality of the upper and lower bounds computed by DFSPE during the exploration of the partial sequences that lead to the optimal solution z^* for instance M1, which is representative of what happens for the other instances. We report: the number of remaining arcs for expansion at period t ($|\hat{\boldsymbol{A}}^{\boldsymbol{t}}|$), the cost of the unsplittable multi-commodity flow solution at period t ($\boldsymbol{z^*(x^t)}$), the average cost per period of the remaining transition periods ($\overline{\boldsymbol{z^*(x^{[t+1,T-1]})}}$), the upper bound on the average cost per period in the remaining transition periods ($\overline{\boldsymbol{UB_{[t+1,T-1]}}}$), the gap, in percentage, between the upper bound and the true costs for the remaining transition periods ($\boldsymbol{G_{UB_{[t+1,T-1]}}(\%)}$), the lower bound on the average cost per period in the remaining transition periods ($\overline{\boldsymbol{LB_{[t+1,T-1]}}}$), and the gap, in percentage, between the lower bound and the true costs for the remaining transition periods ($\boldsymbol{G_{LB_{[t+1,T-1]}}(\%)}$). The results are shown in Table 4.5. We note that the gaps between both the upper and lower bounds and the optimal solution found ($z^* = 6,442.00$) are relatively small throughout the exploration of the partial sequences (0.42%, on average). In particular, when half of the complete sequence was explored, the values of $\boldsymbol{G_{UB_{[t+1,T-1]}}(\%)}$ are all less or equal than 0.05%. The gaps for the lower bounds are, in most part, higher, but still relatively small (1.60%, on average).

Finally, in Table 4.6, we summarize and compare the performances of the best variant of the greedy heuristics (GREEDYCOSTRED), the branch and bound algorithm used in CPLEX (B&B) and DFSPE, where we also report the gap, in percentage, between the best solutions found by the algorithms and the best known solution found amongst all methods

Table 4.5: Upper and lower bounds for instance M1.

t	$\lvert\hat{A}^t\rvert$	$z^*(x^t)$	$\overline{z^*(x^{[t+1,T-1]})}$	$\overline{UB_{[t+1,T-1]}}$	$G_{UB_{[t+1,T-1]}}(\%)$	$\overline{LB_{[t+1,T-1]}}$	$G_{LB_{[t+1,T-1]}}(\%)$
0	10	769.11	715.78	723.49	1.06	690.84	3.48
1	9	751.23	711.35	719.86	1.18	692.18	2.69
2	8	738.10	707.52	717.46	1.38	692.18	2.17
3	7	725.62	704.51	710.78	0.88	692.18	1.75
4	6	717.76	701.86	702.08	0.03	692.18	1.38
5	5	710.58	699.68	699.95	0.04	692.18	1.07
6	4	704.75	697.99	698.35	0.05	692.18	0.83
7	3	701.04	696.46	696.59	0.02	692.19	0.61
8	2	697.90	695.02	695.02	0.00	692.19	0.40
9	1	695.02	-	-	-	-	-

($G_{BKS}(\%)$). The optimal solution for each instance is marked with an asterisk.

Although the costs of the first solutions found by DFSPE are, in some cases, more expensive than the first solutions found by CPLEX, we note that, for the larger instances, DFSPE finds better solutions overall than both CPLEX and GREEDYCOSTRED. Recall that in DFSPE, arcs are selected for expansion based on a pre-defined order following the same policy used in GREEDYCOSTRED (i.e., by the potential largest immediate reduction in flow costs). This explains why many of the first solutions found by DFSPE are the same as the ones found by GREEDYCOSTRED. However, the policies for breaking ties in these methods are different; GREEDYCOSTRED break ties by selecting arcs with the highest size of expansion first, then by largest number of commodity paths using the arc and finally by randomly selecting an arc. DFSPE, on the other hand, break ties by selecting the arc that yields the lowest upper bound on the remaining transition costs. Therefore, we see that some of the first solutions found by DFSPE are better than the ones found by GREEDYCOSTRED. The results also show that the best variant of the greedy heuristics was only able to find the optimal solution for L1, which is the smallest instance in the set. The average gap between the solutions found by GREEDYCOSTRED and the optimal solutions found for the small and medium size instances is of only 0.6%, which is relatively small. Overall, DFPSE outperformed both CPLEX in all but one of the larger instances, both in the quality of the best solutions found and in the number of feasible solutions found.

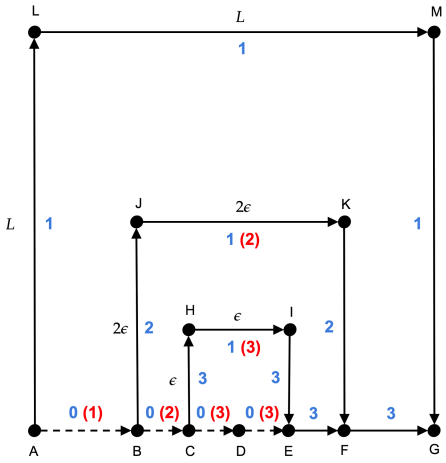
Table 4.6: Computational results for the various algorithms. The best first and overall solutions found are highlighted in bold.

Instance	Algorithm	$\#Sols$	$\overline{z(x)}^1$	$Avg\ \overline{z(x)}$	$\overline{z(x)}^B$	$G_{BKS}(\%)$	TT
S1	GREEDYCOSTRED	-	**63.34***	-	**63.34***	0.00	0.25
	B&B	1	**63.34***	63.34	**63.34***	0.00	0.46
	DFSPE	1	**63.34***	63.34	**63.34***	0.00	1.34
S2	GREEDYCOSTRED	-	158.87	-	158.87	0.56	0.54
	B&B	1	**157.98***	157.98	**157.98***	0.00	0.86
	DFSPE	93	158.58	158.14	**157.98***	0.00	16.59
S3	GREEDYCOSTRED	-	272.27	-	272.27	1.06	1.05
	B&B	1	**269.39***	269.39	**269.39***	0.00	0.94
	DFSPE	29	272.27	270.61	**269.39***	0.00	11.30
S4	GREEDYCOSTRED	-	428.84	-	428.84	0.40	8.03
	B&B	5	**427.26**	427.18	**427.13***	0.00	63.80
	DFSPE	1219	428.84	427.53	**427.13***	0.00	817.60
M1	GREEDYCOSTRED	-	717.23	-	717.23	0.20	17.08
	B&B	8	718.91	716.73	**715.78***	0.00	179.53
	DFSPE	179	**717.23**	716.11	**715.78***	0.00	1,309.14
M2	GREEDYCOSTRED	-	976.00	-	976.00	2.47	76.71
	B&B	18.00	**952.13**	951.87	**951.74***	0.00	82,017.62
	DFSPE	6,663	974.85	968.10	**951.74***	0.00	86,400.00
M3	GREEDYCOSTRED	-	**865.98**	-	865.98	0.45	30.99
	B&B	7	880.20	867.09	**862.11***	0.00	979.96
	DFSPE	5973	**865.98**	864.03	**862.11***	0.00	86,400.00
M4	GREEDYCOSTRED	-	1,165.11	-	1,165.11	0.37	40.97
	B&B	4	**1,161.28**	1,160.32	**1,159.43***	0.00	245.17
	DFSPE	1,265	1,165.08	1,162.21	1,160.82	0.12	86,400.00
L1	GREEDYCOSTRED	-	963.11	-	963.11	0.10	107.69
	B&B	0	-	-	-	-	86,400.00
	DFSPE	3277	**962.86**	962.36	**962.20**	0.00	86,400.00
L2	GREEDYCOSTRED	-	1,115.92	-	1,115.92	0.52	70.01
	B&B	0	-	-	-	-	86,400.00
	DFSPE	1,671	**1,115.27**	1,110.96	**1,110.16**	0.00	86,400.00
L3	GREEDYCOSTRED	-	1,601.57	-	1,601.57	0.13	113.08
	B&B	9	**1,599.96**	1,599.63	**1,599.50***	0.00	13,616.6
	DFSPE	91	1,601.57	1,600.38	**1,599.50***	0.00	86,400.00
L4	GREEDYCOSTRED	-	2,059.09	-	2,059.09	0.02	619.22
	B&B	0	-	-	-	-	86,400.00
	DFSPE	717	**2,059.09**	2,058.87	**2,058.70**	0.00	86,400.00
L5	GREEDYCOSTRED	-	7,232.27	-	7,232.27	0.13	608.88
	B&B	1	7,316.06	7,316.06	7,316.06	1.27	86,400.00
	DFSPE	591	**7,231.98**	7,230.75	**7,223.08**	0.00	86,400.00

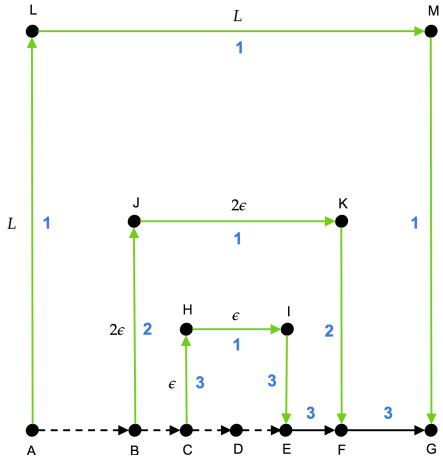
4.7 Incremental network design with temporary arc capacity expansions

In this section, we introduce a variant of the incremental network design problem with multi-commodity flows, in which it is possible to temporarily expand the capacity of arcs, i.e., expand the capacity of arcs that are not are not expanded in the target design for part of the transition period. We will show that using temporary arc expansions, even if it means reaching the target design in more periods (i.e., lengthening the transition period) can result in lower transition costs.

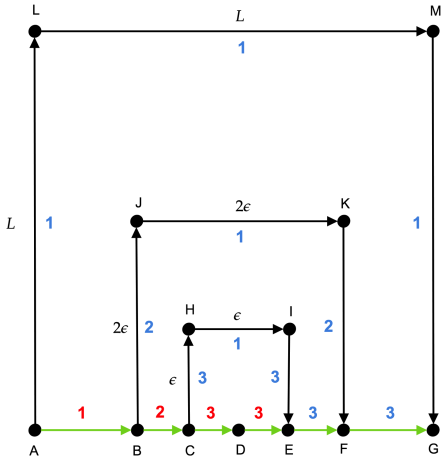
Consider the example shown in Figure 4.7, with three origin-destination pairs with unit demand each (AG, BF and CE) and six candidate arcs for expansion (AB, BC, CD, DE, JK and HI). The initial and target network designs are depicted in Figures 4.7b and 4.7c, with flow costs $3L + 9\epsilon$ and $L + 3\epsilon$, respectively. Observe that only four out of the six candidate arcs for expansion appear with expanded capacity in the target network design (AB, BC, CD and DE). If we can only select a single arc period period and we only select arcs that have expanded capacity on the target design, then the optimal sequence $s_1 =$ (DE, CD, BC, and AB) for a total transition cost of $c(s_1) = (3L+9\epsilon)+(3L+7\epsilon)+(3L+3\epsilon) = 9L+19\epsilon$. On the other hand, if temporary capacity expansion are allowed, then the optimal transition sequence $s_2 =$ (AB, JK, BC, HI, CD, DE) for a total transition cost of $c(s_2) = (3L+9\epsilon)+(L+13\epsilon)+(L+13\epsilon)+(L+9\epsilon)+(L+9\epsilon) = 7L+53\epsilon$. When $L \gg \epsilon$, we have that $c(s_1) \approx c(s_2) + 2L$. In this example, the capacities of the arcs JK and HI are temporarily expanded in order to reduce the total flow costs during the transition periods, and these additional capacities are no longer needed once we expand arcs CD and DE. This scenario shows the potential benefit of considering temporary arc expansions during the transition period.



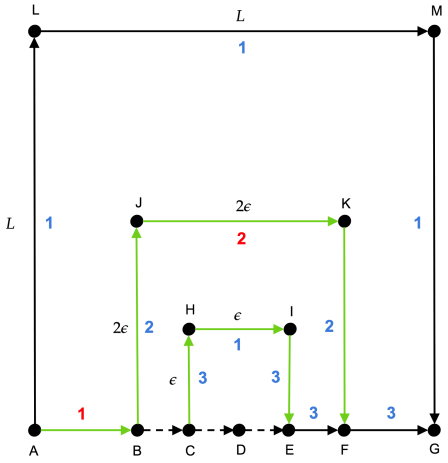
(a) Instance where temporarily expanding the capacity of arcs reduces transition costs



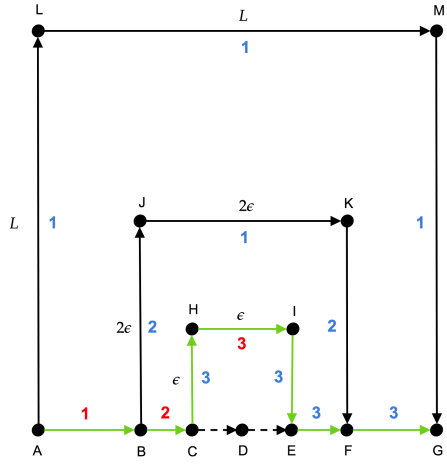
(b) Initial network design



(c) Target network design



(d) Second intermediate design where we temporarily expand the capacity of JK after expanding AB



(e) Fourth intermediate design where we temporarily expand the capacity of HI after expanding AB,JK and BC

Figure 4.7: Example showing the benefit of allowing temporary arc capacity expansions in the network. We consider three origin-destination pairs (AG, BF and CE), six candidate arcs for expansion (AB,BC,CD,DE,JK,HI) from which only four of them appear in the target design.

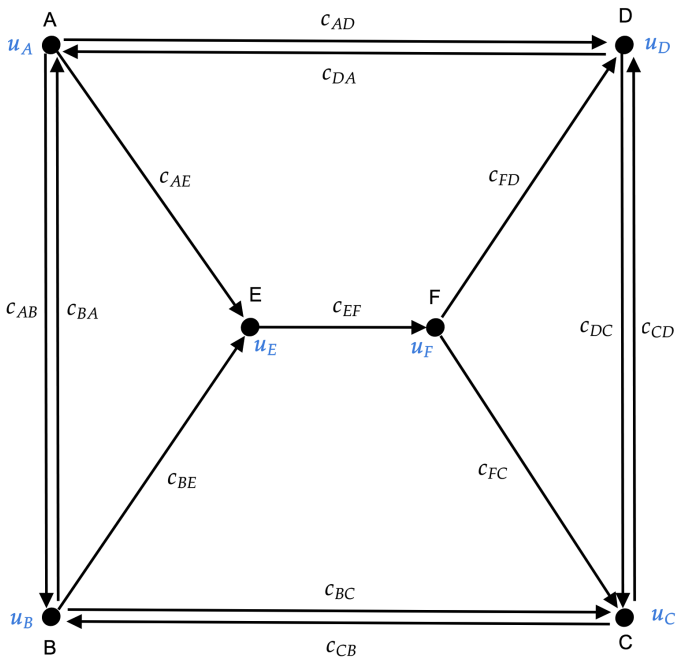
4.8 The hub capacity expansion problem

The algorithms presented in this chapter can be used to solve incremental hub capacity expansion problems for package express transportation carriers using a simple network transformation. Let $G_H^0 = (H, A_H)$ be a graph representing the service network under consideration, where the set node set H represents hubs and the set arcs set A_H represents directed transportation links between hubs. Furthermore, let K be the set of commodities to be served. For each hub $h \in H$, let $u_h \in \mathbb{Z}_+$ be the sorting capacity of the hub, in parcels per day, and for each arc $a \in A_H$, let $c_a \in \mathbb{R}_+$ be the transportation cost of sending one unit of a commodity along the arc. Finally, let G_H^T be a graph defined on the same node and arcs set and $\hat{H} \subseteq H$ represent hubs with expanded capacity $u_h + w_h$, with $w_h \in \mathbb{Z}_+$ the additional sorting capacity of the hub, in parcels per day.

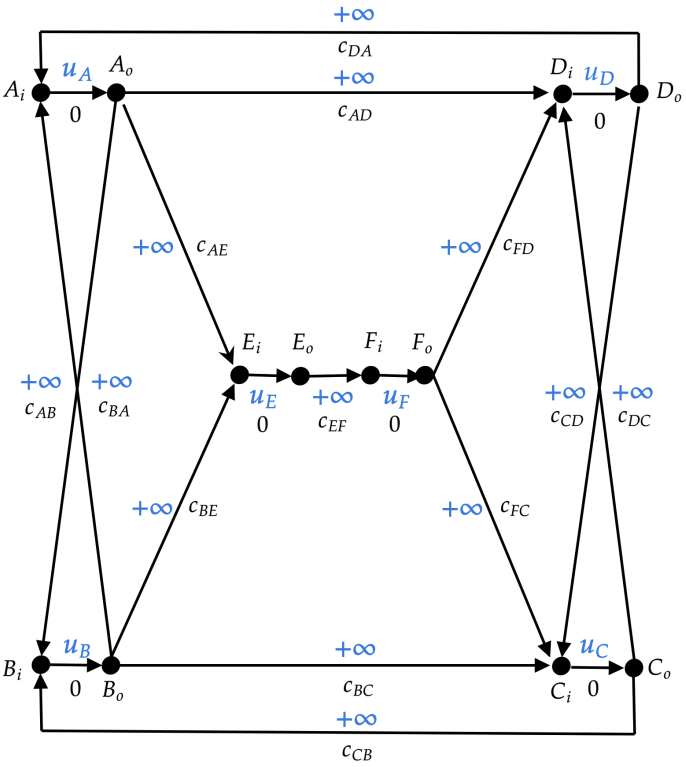
We can transform G_H^0 (and G_H^T) to the form required by the algorithms as follows. First, we will replace every node $h \in H$ by a pair of nodes $\{h_i,h_o\}$ in N, representing an inbound and an outbound node for the hub, respectively, and every arc $a = (n, m) \in A_H$ by an arc $(n_o, m_i) \in A$. Nodes h_i and h_o associated with hub $h \in H$ will be connected by an arc $a = (h_i, h_o) \in A$ with cost zero and capacity $u_a = u_h$. The capacities of all the other arcs in A are set to infinity and the cost of arcs $a = (n_o, m_i)$ is set to be the cost of the arc (n, m) in A_H. Second, we map the origin o_k and destination d_k of each commodity $k \in K$ to $(o_k)_i$ and $(d_k)_i$, respectively. The transformation process from $G_H = (H, A_H)$ to $G = (N, A)$ is illustrated in Figure 4.8 where the set $\hat{A} \subset A$ of arcs with capacity expansions only contains arcs of the form (h_i, h_o), i.e., connecting the inbound and outbound nodes associated with a hub.

4.8.1 Computational study

The proposed algorithms and the transformation process outlined above were used to solve an instance derived from real-world data of SF Express, one of the largest package ex-



(a) Original network $G_H = (H, A_H)$ with nodes representing hubs and arcs representing direct transportation links between hubs



(b) Modifed network $G = (N, A)$ where every node representing a hub $n \in H$ is replaced by two nodes, n_o and n_i, and arcs $(n, m) \in A_H$ are replaced by arcs (n_o, m_i). Pairs of nodes representing the same hub n are connected via an arc (n_i, n_o) with capacity u_n and cost zero.

Figure 4.8: Transformation process from $G_H = (H, A_H)$ to $G = (N, A)$ for the hub capacity expansion problem, with hub capacities in blue and transportation costs in black.

press carriers in China. The data represents demand for multiple service offerings in the southeastern part of China which is to be served by a heterogeneous fleet of vehicles. The demand consists of a set of commodities, each specifying an origin hub, a destination hub, a number of packages, the time the packages are available at the origin hub, and the time the packages are due at the destination hub (which depends on the service offering). Hub sorting capacities are given in parcels per day. Data from a representative day was used to define our instance.

Our computational study involves: (i) using the integer programming model presented in Section 4.6.1 to obtain a target network design in which a small number of hubs are allowed an increase in the sorting capacity to reduce the transportation costs of the commodities, and; (ii) run the proposed algorithms to establish a sequence of hub capacity expansions, where, in each period, the sorting capacity of a single hub is increased and we solve an unsplittable multi-commodity flow problem to find the set of paths for commodities.

Instances

Given that our incremental network design problem with multi-commodity flows does not consider time, we modified the SF Express data as follows. First, we only consider commodities available for pickup during the first day of the planning period. Second, we aggregate commodities with the same origin and destination irrespective of their available and due times. Let the resulting set of commodities be denoted by K_0. Third, using a given load plan, we change the destination of each commodity $k \in K_0$ to the last hub that commodity k visits during the first day. Finally, in order to reduce the size of the instance, we only consider the top 80% hubs in terms of the number of packages available for pickup during the first day. We assume there is a transportation link between every pair of hubs, and set the per-unit cost of sending flow along the link to the average of the per-unit cost of sending flow along the link over the different vehicle types.

Table 4.7 shows the characteristics of: (i) the the SF Express instance (Original); (ii) the extracted instance (Extracted), and; (iii) the incremental version of the extracted instance (E-INC). We report: (i) the number of nodes (hubs) ($|\boldsymbol{N}|$); (ii) the number of arcs (transportation links) ($|\boldsymbol{A}|$), and; (iii) the number of commodities ($|\boldsymbol{K}|$). We generate five different variants of E-INC, in which we randomly select a subset of commodities and either increase or decrease (with equal probability) the number of packages associated with that commodity by $x\%$, where x is selected uniform randomly from [0,20]. The hub capacities for the initial design are taken to be the hub capacities specified in the SF data and the hub capacities in the target design are obtained by solving model TND (Section 4.6.1). Table 4.8 shows the characteristics of the instances generated. We again report: the number of expanded arcs in the target design ($|\hat{\boldsymbol{A}}|$), the average capacity increase in the hubs selected for expansion, in percentage ($\overline{\boldsymbol{w}}(\%)$), the cost of the multi-commodity flow solution on the initial design ($\boldsymbol{z}(\boldsymbol{x^0})$), the cost of the multi-commodity flow solution on the target design ($\boldsymbol{z}(\boldsymbol{x^T})$) and the total time solving the unsplittable multi-commodity flow problem on the the initial network design, in seconds ($\boldsymbol{TT}_{\boldsymbol{z}(\boldsymbol{x^0})}$).

Table 4.7: Original real-world instance and the reduced and adapted instances for the hub capacity expansion problem

Instance	$\|\boldsymbol{N}\|$	$\|\boldsymbol{A}\|$	$\|\boldsymbol{K}\|$
Original	70	4830	115,115
Extracted	56	3080	795
E-INC	112	3136	795

Analysis

Table 4.9 reports the following statistics for the solutions produced by the greedy heuristics: the average cost of the multi-commodity flow solution per period ($\overline{\boldsymbol{z}(\boldsymbol{x})}$), the cost of the unsplittable multi-commodity flow solutions on the first and last intermediate designs ($\boldsymbol{z}(\boldsymbol{x^1})$ and $\boldsymbol{z}(\boldsymbol{x^{T-1}})$, respectively), and the total running time of the algorithm ($\boldsymbol{TT}$).

Table 4.8: Instances generated for the hub capacity expansion problem

Instance	$\lvert\hat{A}\rvert$	$\overline{w}(\%)$	$z(x^0)$	$z(x^T)$	$TT_{z(x^0)}$
E-INC-1	6	62.11	942,117.39	932,070.38	160.63
E-INC-2	8	59.57	908,498.15	898,226.81	104.02
E-INC-3	10	51.60	910,662.98	902,293.62	107.66
E-INC-4	10	57.31	908,353.05	899,830.06	45.96
E-INC-5	10	45.48	908,498.15	898,215.69	99.42

Table 4.9: Computational results for the greedy heuristics on the set of instances generated for the hub capacity expansion problem. The best solutions are highlighted in bold.

Instance	**Algorithm**	$z(x^1)$	$z(x^{T-1})$	$\overline{z(x)}$	$\overline{TT}$	TT
E-INC-1	GREEDYSIZEEXP-LH	941,755.71	932,379.32	938,160.90	171.41	857.06
	GREEDYSIZEEXP-HL	941,606.33	932,425.49	**935,938.40**	239.83	1,199.13
	GREEDYCOSTRED	941,606.33	932,425.49	**935,938.40**	262.84	1,314.19
E-INC-2	GREEDYSIZEEXP-LH	908,475.58	898,210.51	904,738.79	108.35	758.46
	GREEDYSIZEEXP-HL	908,498.15	898,331.66	902,007.50	144.67	1,012.71
	GREEDYCOSTRED	906,490.39	898,257.34	**900,528.79**	141.76	992.35
E-INC-3	GREEDYSIZEEXP-LH	910,621.13	902,293.48	908,110.89	149.67	1,347.01
	GREEDYSIZEEXP-HL	910,662.98	902,415.23	904,655.56	123.37	1,110.36
	GREEDYCOSTRED	906,385.21	902,332.33	**903,276.11**	154.82	1,393.35
E-INC-4	GREEDYSIZEEXP-LH	908,262.50	900,141.77	905,036.67	119.85	1,078.69
	GREEDYSIZEEXP-HL	908,340.50	899,932.24	**902,947.00**	145.26	1,307.30
	GREEDYCOSTRED	908,095.60	899,924.05	904,628.33	122.91	1,106.20
E-INC-5	GREEDYSIZEEXP-LH	908,475.58	900,232.48	905,000.06	107.70	969.30
	GREEDYSIZEEXP-HL	908,498.15	898,299.71	901,539.67	107.86	970.74
	GREEDYCOSTRED	906,490.39	898,261.71	**900,077.17**	120.73	1,086.61

We observe that, once more, GREEDYCOSTRED outperformed the greedy heuristics by size of expansion for the majority of the instances, finding the best solution in four out of of the five instances. In Table 4.10, we report the following statistics of the sequence of expansions determined by GREEDYCOSTRED for instance E-INC-5: the original hub sorting capacity, in parcels per day, of the hub selected for expansion at period t (u_{h_t}), the relative size of the hub selected for expansion at period t, in percentage ($w_{h_t}(\%)$), the cost of the unsplittable multi-commodity flow solution at period t ($z(x^t)$), and the average cost per period of the remaining transition periods ($\overline{z(x^{[t+1,T-1]})}$). We observe that the largest reductions in flow cost happen in the first iterations (0.22% from period 0 to period 1 and 0.47% from period 1 to period 2), followed by very small reductions from period 2 onwards (0.05%, on average). It is also interesting to observe that the hubs with the smallest and largest sizes of expansion are selected at the very last periods, which strengthens the case that expanding capacity of hubs (arcs) purely based on size of expansion may result in bad solutions.

Table 4.10: Hub capacity expansions and flow costs at each period for E-INC-5

t	u_{h_t}	$w_{h_t}(\%)$	$z(x^t)$	$\overline{z(x^{[t+1,T-1]})}$
0	-	-	908,498.15	900,077.17
1	552,000	30	906,490.39	899,275.51
2	432,000	50	902,189.99	898,859.16
3	360,000	45	900,278.57	898,622.59
4	172,800	45	899,513.35	898,444.44
5	86,400	50	898,643.31	898,394.72
6	156,000	50	898,643.31	898,311.86
7	36,000	50	898,336.84	898,299.37
8	312,000	45	898,336.84	898,261.71
9	7,200	50	898,261.71	0.00
10	720,000	50	898,215.69	-

4.9 Final Remarks

More practical aspects and constraints need to be incorporated in the model presented in this chapter to be able to develop effective real-world decision support for package express carriers. A more thorough investigation of temporary capacity expansions (Section 4.7) may also be needed and beneficial before developing practical decision support tools to better understand the trade-offs between operating profits and the timing and size of investments in additional capacity.

Appendices

APPENDIX A

FAST AND ROBUST ALGORITHMS FOR SOLVING BOMIPS

A.1 Pseudo-codes

Below, we present the pseudo-codes for the following algorithms: (i) PURELEX (Algorithm 8); (ii) the line generation procedured used in PURELEX (Algorithm 9); (iii) SPURELEX, Phase 1 (Algorithm 10) and Phase 2 (Algorithm 11). We do not include the pseudo-code for the enhanced implementation of the ϵ-Tabu Constraint Method since it is a simple modification of the original pseudo-code presented in [20].

Algorithm 8 PURELEX

input: L - upper left corner of the first box
R - bottom right corner of the first box
output: N - non dominated frontier

$Q \leftarrow B(L, R)$ # initial region defined by box L, R
while $Q \neq \emptyset$ **do**
  $B(z^L, z^R) \leftarrow element(Q)$
  $Q \leftarrow setminus(Q, B(z^L, z^R))$
  $\mu \leftarrow (z_2^L, z_2^R)/2$ # horizontal line between z^L, z^R
  $z^* \leftarrow \text{lexmin}\{(z_1, z_2) : z_2(x) \leq \mu\}$ # finds NDP in lower half
  if $\mu - z_2^* > \epsilon$ **then**
    $Q \leftarrow Q \cup B(z^*, z^R)$ # adds a new region to the queue
    $\hat{z} \leftarrow \text{lexmin}\{(z_2, z_1) : z_1(x) \leq z_1^* - \epsilon\}$
    $Q \leftarrow Q \cup B(z^L, \hat{z})$
  else
    $(\hat{z}^1, \hat{z}^2, z^1, z^2, M) \leftarrow LineRestriction(z^*, z^L, z^R)$
    $N \leftarrow N \cup M \cup line(z^1, z^2)$
    $Q \leftarrow Q \cup B\left(z^L, \hat{z}^1\right)$ # adds to queue
    $Q \leftarrow Q \cup B\left(\hat{z}^2, z^R\right)$ # adds to queue
  end if
end while
return N

Algorithm 9 LineRestriction

input: z^* - NDP in the lower half region of the box
L - upper left corner of the box
R - bottom right corner of the box
output: non dominated portion of the line segment

1: $(z^1, z^2, \vec{w}^T, w\_known) \leftarrow$ LineGen(z^*, L, R)
2: $M \leftarrow \emptyset$
3: $z^\alpha \leftarrow z^1, z^\beta \leftarrow z^2$
4: **if** w_known **then**
5: $z^\alpha \leftarrow$ lexmin$(z_2(x), z_1(x))$
6: **if** $feasible$ **then**
7: $M \leftarrow M \cup \{z^\alpha\}$
8: $z^1 \leftarrow horizontalProjection(z^\alpha, Line(z^1, z^*))$
9: $z1\_open \leftarrow true$
10: **end if**
11: $z^\beta \leftarrow$ lexmin$(z_1(x), z_2(x))$
12: **if** $feasible$ **then**
13: $M \leftarrow M \cup \{z^\beta\}$
14: $z^2 \leftarrow verticalProjection(z^\beta, Line(z^*, z^2))$
15: $z2\_open \leftarrow true$
16: **end if**
17: **end if**
18: **return** $(z^\alpha, z^\beta, z^1, z^2, M)$

Algorithm 10 SPURELEX: Phase One

input: L - upper left corner of the first box
R - bottom right corner of the first box
output: N - non dominated frontier

$Q \leftarrow B(L, R)$ # initial region defined by box L, R
$totalVolume \leftarrow volume(B(L, R))$
$unsolved \leftarrow totalVolume$
while $Q \neq \emptyset$ **do**
  $B(z^L, z^R) \leftarrow element(Q)$ # gets box with largest volume
  if $(unsolved - volume(B(z^L, z^R)))/totalVolume \leq \rho$ **then**
    $N \leftarrow N \cup$ PhaseTwo(Q)
    return N
  end if
  $Q \leftarrow Q \setminus \{B(z^L, z^R)\}$
  $unsolved \leftarrow unsolved - volume(B(z^L, z^R))$
  $\mu \leftarrow (z_2^L, z_2^R)/2$ # horizontal line between z^L, z^R
  $z^* \leftarrow$ lexmin$\{(z_1, z_2) : z_2(x) \leq \mu\}$ # finds NDP in lower half
  if $\mu - z_2^* > \epsilon$ **then**
    $Q \leftarrow Q \cup B(z^*, z^R)$ # adds a new region to the queue
    $\hat{z} \leftarrow$ lexmin$\{(z_2, z_1) : z_1(x) \leq z_1^* - \epsilon\}$
    $Q \leftarrow Q \cup B(z^L, \hat{z})$
  else
    $(\hat{z}^1, \hat{z}^2, z^1, z^2, M) \leftarrow LineRestriction(z^*, z^L, z^R)$
    $N \leftarrow N \cup M \cup line(z^1, z^2)$
    $Q \leftarrow Q \cup B\left(z^L, \hat{z}^1\right)$ # adds to queue
    $Q \leftarrow Q \cup B\left(\hat{z}^2, z^R\right)$ # adds to queue
  end if
end while
return N

Algorithm 11 SPURELEX: Phase Two

input: Q - stack with remaining boxes to process
output: final portion of the non dominated frontier

$N' \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  $B(z^L, z^R) \leftarrow element(Q)$
  $Q \leftarrow Q \setminus \{B(z^L, z^R)\}$
  $N' \leftarrow N' \cup \epsilon$-TM($B(z^L, z^R)$) # finds NDF with ϵ-TM or enhanced ϵ-TM
end while
return N'

A.2 Instance generation

Here we elaborate on the details of the generation of the bent instances. The nondominated frontier is bounded within $z_i(x) \in [-k, k]$ for $i = 1, 2$, and we choose large enough k, e.g. $k = 1.5 * (n + 1)$, in order to avoid numerical issues. We define the "unbent" line segment as $L = \{(x_1, x_2) \in \mathbb{R}^2 : x_1 + x_2 = 0, -k \leq x_i \leq k \text{ for } i = 1, 2\}$. The NDPs are chosen from a line segment that is shifted downward from L by d, and from left to right, an NDP is chosen a horizontal distance of $2d + 1$ away from the previous NDP in order to avoid the cones from simultaneously dominating a portion of L (this only holds for bounding θ_1, θ_2 as done in the next step).

Algorithm 12 Randomized Cone-Width NDP Generation

1: $d = k/(n + 1) - 0.5$
2: $a_1 = -k + 0.5d$
3: $b_1 = -a_1 - d$
4: **for** $i = 2, ..., n$ **do**
5: $\quad a_i = a_{i-1} + 2d + 1$
6: $\quad b_i = -a_i - d$
7: **end for**

Given corner point (a, b), a cone in objective space is generally defined by $\{z \in \mathbb{R}^2 : \theta_1 z_1 + (1 - \theta_1) z_2 \geq \theta_1 a + (1 - \theta_1) b, \theta_2 z_1 + (1 - \theta_2) z_2 \geq \theta_2 a + (1 - \theta_2) b\}$, where $\theta_1 \in [\frac{3}{4}, 1]$ and $\theta_2 \in [0, \frac{1}{4}]$. Let π be the probability that a cone is orthogonal (i.e., $\theta_1 = 1$ and $\theta_2 = 0$); we use $\pi = 0.05$.

Algorithm 13 Randomized Theta Generation

thetalist= $\emptyset$
for $i = 1, 2, ..., n$ **do**
  if $U(0,1) \leq \pi$ **then**
    $\theta_1 = 1$
    $\theta_2 = 0$
  else
    $\theta_1 = U(\frac{3}{4}, 1)$
    $\theta_2 = U(0, \frac{1}{4})$
  end if
  thetalist.append((θ_1, θ_2))
end for

The "bent" line segment has corner point $(a_0, b_0) = (-d/4, -d/4)$, which may be dominated or nondominated, $\theta_1^0 = (k + d/4)/(2k)$, and $\theta_2^0 = (k - d/4)/(2k)$. Then the bent line segment $\hat{L}$ fits the previous definition for a cone while substituting the corner point (a_0, b_0), θ_1^0, and θ_2^0. Let the generated NDPs be $\{(a_i, b_i)\}_{i=1,2,...,n}$ and their cones be defined by $\{(\theta_1^i, \theta_2^i)\}_{i=1,2,...,n}$. We then have the following BOMILP for the bent instance:

$$
\begin{aligned}
&\min \quad (x_1, x_2) \\
&\text{s.t.} \quad \theta_1^i x_1 + (1 - \theta_1^i) x_2 \geq \theta_1^i a_i + (1 - \theta_1^i) b_i - 2k(1 - y_i) \; \forall i = 0, ..., n \\
&\theta_2^i x_1 + (1 - \theta_2^i) x_2 \geq \theta_2^i a_i + (1 - \theta_2^i) b_i - 2k(1 - y_i) \; \forall i = 0, \ldots, n \\
&\sum_{i=0}^{n} y_i = 1 \\
&-k \leq x_i \leq k \; \forall i = 1, 2 \\
&y \in \{0, 1\}^{n+1}
\end{aligned}
$$

A.3 Approximation results

Below, we present the numerical results of the approximation of the NDF for ϵTM (Table A.1), for PURELEX (Table A.2), for SPURELEX (Table A.3), and for the recursive variant of BLM (Table A.4). We report the following statistics: **TP**, the point in time (during the execution of the algorithm) that the statistics were collected; **fINDP**, the fraction of the number of isolated NDPs found; **fNLS**, the fraction of the number of nondominated line segments found; **fLNLS**, the fraction of the total length of nondominated line segments found; **fA**, the resolved area of the initial box $B(z^L, z^R)$ as a fraction; and, **fSlice**, the fraction of slices that contribute to the frontier found. When not applicable, an entry in the tables is marked with "-".

Table A.1: Approximation results for ϵTM.

Instance	ST	TP	M1	fINDP	fNLS	fTNLS	fA	fSlice
21	16.26	16.26	99.582673%	-	0.24%	0.30%	0.42%	0.68%
	34.10	34.10	99.433120%	-	0.33%	0.38%	0.57%	0.68%
	68.34	68.34	97.547610%	-	0.66%	0.77%	2.45%	1.02%
	128.22	128.22	93.596498%	-	1.74%	1.97%	6.40%	2.03%
	263.72	263.73	89.951496%	-	3.33%	3.52%	10.05%	3.05%
	512.45	512.46	85.318206%	-	5.88%	6.46%	14.68%	5.42%
	1024.26	1024.29	77.467321%	-	11.02%	11.19%	22.53%	9.83%
	2057.03	2057.08	63.777420%	-	20.82%	20.08%	36.22%	18.31%
	4096.58	4096.68	40.377154%	-	38.86%	37.06%	59.62%	34.58%
	8192.29	8192.50	2.248251%	-	82.51%	85.20%	97.75%	81.36%
	9690.87	9691.13	0.000000%	-	100.00%	100.00%	100.00%	100.00%
Bent7500.A	16.77	16.77	99.977826%	0.55%	0.02%	0.01%	0.02%	0.03%
	34.54	34.54	99.909084%	0.55%	0.05%	0.04%	0.09%	0.05%
	64.12	64.12	99.800146%	0.55%	0.11%	0.10%	0.20%	0.11%
	130.20	130.21	99.575419%	0.82%	0.22%	0.21%	0.42%	0.23%
	258.36	258.37	99.152935%	1.64%	0.44%	0.41%	0.85%	0.44%
	514.80	514.82	98.225619%	2.46%	0.90%	0.88%	1.77%	0.91%
	1027.33	1027.38	96.304858%	3.01%	1.88%	1.88%	3.70%	1.88%
	2049.65	2049.76	92.163824%	5.19%	4.01%	4.05%	7.84%	4.01%
	4099.85	4100.13	83.302600%	9.02%	8.74%	8.87%	16.70%	8.75%
	8192.22	8192.96	63.073380%	18.58%	20.59%	20.86%	36.93%	20.60%
	16384.52	16386.97	10.473017%	66.12%	67.65%	67.40%	89.53%	67.64%
	17977.38	17981.29	0.000000%	100.00%	100.00%	100.00%	100.00%	100.00%
Rand7500.A	18.85	18.85	99.979689%	0.27%	0.02%	0.01%	0.02%	0.03%
	33.15	33.15	99.953675%	0.27%	0.04%	0.02%	0.05%	0.04%
	70.83	70.83	99.887789%	0.27%	0.07%	0.06%	0.11%	0.07%
	131.37	131.37	99.768739%	0.27%	0.13%	0.12%	0.23%	0.13%
	259.62	259.62	99.539288%	0.55%	0.24%	0.23%	0.46%	0.24%
	512.96	512.97	99.061224%	0.55%	0.48%	0.47%	0.94%	0.48%
	1028.21	1028.23	98.098272%	1.09%	0.96%	0.96%	1.90%	0.97%
	2050.24	2050.29	96.153765%	1.37%	1.96%	1.96%	3.85%	1.96%
	4100.09	4100.20	92.280887%	3.55%	3.95%	3.95%	7.72%	3.95%
	8195.05	8195.32	84.232517%	10.66%	8.23%	8.18%	15.77%	8.24%
	16385.06	16385.68	65.910199%	23.22%	18.82%	18.74%	34.09%	18.82%
	32768.88	32771.07	18.845577%	59.02%	56.60%	56.54%	81.15%	56.61%
	35789.02	35793.24	0.000000%	100.00%	100.00%	100.00%	100.00%	100.00%

Table A.2: Approximation results for PURELEX.

Instance	ST	TP	M1	fINDP	fNLS	fTNLS	fA	fSlice
21	16.22	16.22	2.060236%	-	0.10%	0.22%	89.76%	3.06%
	36.15	36.15	0.694905%	-	0.20%	0.32%	94.89%	5.78%
	65.88	65.88	0.165257%	-	0.39%	0.49%	97.41%	10.88%
	128.38	128.38	0.043796%	-	0.79%	0.98%	98.73%	21.43%
	257.16	257.17	0.010580%	-	1.53%	1.94%	99.38%	39.80%
	513.99	514.00	0.002459%	-	3.86%	5.01%	99.71%	65.99%
	1024.45	1024.47	0.000749%	-	14.03%	15.68%	99.86%	80.61%
	2049.89	2049.96	0.000172%	-	43.22%	46.53%	99.96%	89.80%
	4100.56	4100.71	0.000011%	-	82.93%	87.12%	100.00%	96.94%
	6777.20	6777.46	0.000000%	-	100.00%	100.00%	100.00%	100.00%
Bent7500.A	16.00	16.01	0.389899%	0.27%	0.19%	0.13%	94.74%	0.11%
	32.01	32.02	0.024330%	1.64%	0.71%	0.46%	98.62%	0.47%
	64.08	64.11	0.006002%	2.46%	1.78%	1.07%	99.46%	1.45%
	128.03	128.10	0.001483%	5.46%	3.94%	2.89%	99.77%	1.95%
	256.12	256.30	0.000348%	8.74%	8.41%	5.09%	99.90%	6.76%
	512.01	512.40	0.000074%	19.40%	17.09%	12.59%	99.96%	8.32%
	1024.16	1025.00	0.000016%	43.44%	34.72%	27.62%	99.99%	11.81%
	2048.14	2049.95	0.000002%	86.07%	68.92%	58.97%	100.00%	15.17%
	4096.22	4100.11	0.000000%	100.00%	100.00%	85.28%	100.00%	100.00%
	5442.12	5447.43	0.000000%	100.00%	100.00%	100.00%	100.00%	100.00%
Rand7500.A	16.06	16.06	0.097490%	0.27%	0.35%	0.20%	97.28%	0.27%
	32.26	32.27	0.024260%	1.37%	0.87%	0.54%	98.87%	0.55%
	64.25	64.29	0.005985%	1.64%	1.94%	1.09%	99.51%	1.68%
	128.19	128.27	0.001476%	3.83%	4.10%	2.55%	99.78%	2.53%
	256.19	256.38	0.000348%	8.47%	8.48%	4.77%	99.90%	7.28%
	512.00	512.44	0.000080%	17.76%	17.16%	10.81%	99.96%	10.05%
	1024.18	1025.11	0.000016%	36.89%	34.60%	22.61%	99.98%	16.37%
	2048.03	2049.93	0.000002%	74.59%	69.34%	46.47%	100.00%	26.13%
	4096.02	4099.76	0.000000%	100.00%	100.00%	79.49%	100.00%	100.00%
	5420.87	5426.26	0.000000%	100.00%	100.00%	100.00%	100.00%	100.00%

Table A.3: Approximation results for SPURELEX with $\rho = 0.005$.

Instance	ST	TP	M1	fINDP	fNLS	fTNLS	fA	fSlice
21	16.57	16.57	2.060236%	-	0.10%	0.22%	89.76%	3.05%
	32.46	32.46	0.752281%	-	0.19%	0.31%	94.51%	5.42%
	65.80	65.80	0.173392%	-	0.38%	0.48%	97.33%	10.51%
	129.53	129.53	0.048528%	-	0.75%	0.90%	98.65%	20.34%
	256.90	256.90	0.012639%	-	1.44%	1.85%	99.34%	37.97%
	512.32	512.33	0.006956%	-	4.45%	5.37%	99.51%	48.81%
	1024.33	1024.36	0.006956%	-	9.61%	10.29%	99.53%	51.19%
	2048.02	2048.08	0.006956%	-	21.08%	22.94%	99.58%	54.92%
	4096.39	4096.52	0.006956%	-	42.20%	43.49%	99.66%	66.10%
	8192.23	8192.51	0.006956%	-	89.11%	88.52%	99.91%	91.19%
	9120.40	9120.72	0.000000%	-	100.00%	100.00%	100.00%	100.00%
Bent7500.A	16.07	16.07	0.389944%	0.27%	0.16%	0.12%	94.16%	0.07%
	32.07	32.08	0.024330%	1.64%	0.66%	0.42%	98.56%	0.47%
	64.24	64.27	0.006009%	2.46%	1.67%	1.00%	99.42%	1.37%
	128.10	128.27	0.005985%	4.92%	4.92%	4.13%	99.51%	4.61%
	256.09	256.57	0.005985%	12.30%	11.79%	11.05%	99.54%	11.64%
	512.00	513.25	0.005985%	28.42%	25.53%	24.91%	99.59%	25.50%
	1024.00	1026.81	0.005985%	56.56%	52.93%	52.62%	99.70%	52.73%
	1907.78	1913.49	0.000000%	100.00%	100.00%	100.00%	100.00%	100.00%
Rand7500.A	16.13	16.14	0.097490%	0.27%	0.35%	0.20%	97.28%	0.27%
	32.05	32.07	0.024260%	1.37%	0.86%	0.53%	98.85%	0.55%
	64.01	64.04	0.005986%	1.64%	1.92%	1.08%	99.50%	1.67%
	128.01	128.19	0.005986%	6.01%	5.46%	4.62%	99.52%	5.28%
	256.01	256.52	0.005986%	13.66%	12.47%	11.68%	99.54%	12.36%
	512.00	513.31	0.005986%	25.14%	26.22%	25.62%	99.60%	26.26%
	1024.10	1027.19	0.005986%	51.64%	54.05%	53.79%	99.71%	53.83%
	1866.35	1872.18	0.000000%	100.00%	100.00%	100.00%	100.00%	100.00%

Table A.4: Approximation results for the recursive variant of BLM.

Instance	ST	TP	M1	fINDP	fNLS	fTNLS	fA	fSlice
21	18.03	18.03	1.209597%	-	0.14%	0.25%	92.50%	4.08%
	32.76	32.76	0.694905%	-	0.22%	0.34%	94.89%	6.12%
	65.46	65.46	0.149381%	-	0.42%	0.52%	97.50%	11.56%
	129.23	129.23	0.070827%	-	0.73%	0.77%	98.29%	17.01%
	257.06	257.06	0.014935%	-	1.53%	1.81%	99.27%	37.76%
	512.90	512.90	0.003389%	-	3.01%	3.58%	99.66%	62.59%
	1056.77	1056.78	0.000906%	-	12.11%	13.85%	99.84%	79.59%
	2049.07	2049.14	0.000249%	-	38.11%	41.63%	99.95%	90.14%
	4096.38	4096.55	0.000016%	-	80.31%	84.72%	100.00%	98.30%
	6482.32	6482.60	0.000000%	-	100.00%	100.00%	100.00%	100.00%
Rand7500.A	1151.45	1151.47	25.000000%	0.27%	0.76%	0.55%	68.98%	0.93%
	1835.93	1835.96	4.246974%	0.27%	0.87%	0.60%	87.42%	1.09%
	1916.55	1916.61	4.028313%	1.37%	2.55%	1.95%	90.50%	3.05%
	2042.68	2043.04	1.496833%	3.55%	6.11%	4.87%	93.50%	7.16%
	2071.65	2071.84	1.256458%	3.55%	6.19%	4.91%	94.50%	7.28%
	2096.62	2096.83	1.061497%	3.55%	6.26%	4.94%	95.34%	7.37%
	2119.79	2120.00	1.006409%	3.55%	6.34%	4.98%	96.05%	7.49%
	2132.32	2132.56	0.773759%	3.55%	6.43%	5.02%	96.74%	7.61%
	4096.03	4098.84	0.000016%	44.54%	64.17%	51.40%	99.99%	75.06%
	5821.14	5826.61	0.000000%	100.00%	100.00%	100.00%	100.00%	100.00%

APPENDIX B

HEURISTICS FOR THE SAME-DAY DELIVERY PROBLEM WITH HUB CAPACITY CONSTRAINTS

We report the results summarizing the performance of MILS for different values of η on instance B, i.e. $\eta = 0$ (Tables B.1 and B.4), $\eta = 0.005$ (Tables B.2 and B.5) and $\eta = 0.05$ (Tables B.3 and B.6), with $It^O_{max} = 10, It^I_{max} = 500, T_k = -1, p = 0.05, It^I_R = 100$. The results are representative of what happens for the other instances. In Tables B.1-B.3, we report the following statistics: the iteration of the outer loop ($\boldsymbol{It^O}$), the number of commodities served in the initial solution generated by the greedy heuristic ($\boldsymbol{K^{S_0}}$), the number of commodities served in the best local solution found ($\boldsymbol{K^S}$), the cost of the best local solution found ($\boldsymbol{C}$), and the total running time of the iteration (**TT**). In Tables B.4-B.6, we report: how many times the neighborhood was called, in percentage ($\boldsymbol{N(\%)}$), in how many iterations the neighborhood managed to modify the solution either increasing, decreasing or preserving the number of commodities served, in percentage ($\boldsymbol{M(\%)}$), in how many iterations, out of $\boldsymbol{M(\%)}$, the neighborhood improved the solution in terms of the number of commodities served immediately after the move was applied, in percentage ($\boldsymbol{I_M(\%)}$), in how many iterations, out of $\boldsymbol{M(\%)}$, the neighborhood deteriorated the solution in terms of the number of commodities served immediately after the move was applied, in percentage ($\boldsymbol{D_M(\%)}$), and in how many iterations, out of $\boldsymbol{M(\%)}$, the neighborhood preserved the same number of commodities served immediately after the move was applied, in percentage ($\boldsymbol{P_M(\%)}$)

Table B.1: MILS in-depth analysis of the 10 iterations of the outer loop for instance B with $\eta = 0$

It^O	$\lvert K^{S_0}\rvert$	$\lvert K^S\rvert$	C	TT
1	760	946	11802.61	438.64
2	766	953	11381.34	425.21
3	780	947	12358.31	250.32
4	775	948	12058.09	295.48
5	772	945	12099.44	519.35
6	761	939	12279.42	485.92
7	762	945	12221.96	458.99
8	769	947	12656.78	279.89
9	777	947	12663.58	238.41
10	788	943	13152.33	311.47
Avg.	771.00	946.00	12267.38	370.36

Table B.2: MILS in-depth analysis of the 10 iterations of the outer loop for instance B with $\eta = 0.005$

It^O	$\lvert K^{S_0}\rvert$	$\lvert K^S\rvert$	C	TT
1	769	952	11866.31	573.53
2	766	952	12281.38	648.43
3	772	960	11725.98	510.59
4	773	950	12442.57	460.03
5	785	956	11840.96	738.07
6	783	955	12496.56	556.83
7	780	955	11787.88	650.85
8	775	955	12648.75	497.97
9	762	953	12124.19	615.10
10	779	951	12321.79	467.00
Avg.	774.40	953.90	12153.63	571.84

Table B.3: MILS in-depth analysis of the 10 iterations of the outer loop for instance B with $\eta = 0.05$

It^O	$\|K^{S_0}\|$	$\|K^S\|$	C	TT
1	770	946	12319.48	643.09
2	771	943	12728.59	433.22
3	769	940	12649.73	454.98
4	762	936	12897.37	422.06
5	759	940	13476.15	418.28
6	761	938	13312.00	426.45
7	761	932	13504.02	381.15
8	770	942	13319.10	422.22
9	772	942	13552.02	376.65
10	761	937	12851.54	380.74
Avg.	765.60	939.60	13061.00	435.88

Table B.4: MILS in-depth analysis showing the average performance of the several neighborhoods for instance B with $\eta = 0$.

Neighborhood	**N(%)**	**M(%)**	$I_M(\%)$	$D_M(\%)$	$P_M(\%)$
MOVELOADINGPERIOD	9.11	< 0.01	65.41	4.59	30.00
MOVEUNLOADINGPERIOD	9.07	< 0.01	74.61	5.39	20.00
CHANGELOADINGDOCK	9.10	< 0.01	0.00	0.00	100.00
CHANGEUNLOADINGDOCK	9.08	< 0.01	0.00	0.00	100.00
SWITCHLOADINGPERIOD	9.09	0.39	30.67	9.33	60.00
RE-ARRANGELOADINGPERIODS	9.09	< 0.01	100.00	0.00	0.00
RE-ARRANGEUNLOADINGPERIODS	9.10	< 0.01	100.00	0.00	0.00
SWITCHDIRECTPATH	9.08	1.62	0.00	4.21	95.79
OPENNEWLOADINGPERIOD	9.10	0.21	100.00	0.00	0.00
OPENNEWUNLOADINGPERIOD	9.11	0.60	100.00	0.00	0.00
CROSSOVERPATHS	9.07	< 0.01	0.00	0.00	100.00

Table B.5: MILS in-depth analysis showing the average performance of the several neighborhoods for instance B with $\eta = 0.005$.

Neighborhood	**N(%)**	**M(%)**	$I_M(\%)$	$D_M(\%)$	$P_M(\%)$
MOVELOADINGPERIOD	9.16	< 0.01	61.41	6.99	31.60
MOVEUNLOADINGPERIOD	9.16	< 0.01	58.52	7.71	33.77
CHANGELOADINGDOCK	9.14	< 0.01	0.00	0.00	100.00
CHANGEUNLOADINGDOCK	9.14	< 0.01	0.00	0.00	100.00
SWITCHLOADINGPERIOD	8.73	0.37	29.47	7.05	63.48
RE-ARRANGELOADINGPERIODS	9.16	< 0.01	100.00	0.00	0.00
RE-ARRANGEUNLOADINGPERIODS	9.18	< 0.01	100.00	0.00	0.00
SWITCHDIRECTPATH	6.92	1.46	0.00	2.00	98.00
OPENNEWLOADINGPERIOD	9.61	0.17	100.00	0.00	0.00
OPENNEWUNLOADINGPERIOD	10.63	0.48	100.00	0.00	0.00
CROSSOVERPATHS	9.17	< 0.01	0.00	0.00	100.00

Table B.6: MILS in-depth analysis showing the average performance of the several neighborhoods for instance B with $\eta = 0.05$.

Neighborhood	**N(%)**	**M(%)**	$I_M(\%)$	$D_M(\%)$	$P_M(\%)$
MOVELOADINGPERIOD	8.72	< 0.01	61.11	8.89	30.00
MOVEUNLOADINGPERIOD	8.72	< 0.01	55.78	4.22	40.00
CHANGELOADINGDOCK	8.69	< 0.01	0.00	0.00	100.00
CHANGEUNLOADINGDOCK	8.68	< 0.01	0.00	0.00	100.00
SWITCHLOADINGPERIOD	7.18	0.31	36.41	13.59	50.00
RE-ARRANGELOADINGPERIODS	8.77	< 0.01	100.00	0.00	0.00
RE-ARRANGEUNLOADINGPERIODS	8.78	< 0.01	100.00	0.00	0.00
SWITCHDIRECTPATH	2.62	1.78	0.00	4.78	95.22
OPENNEWLOADINGPERIOD	10.47	0.05	100.00	0.00	0.00
OPENNEWUNLOADINGPERIOD	18.67	0.11	100.00	0.00	0.00
CROSSOVERPATHS	8.69	< 0.01	0.00	0.00	100.00

REFERENCES

[1] R. Agarwal and Ö. Ergun, "Ship scheduling and network design for cargo routing in liner shipping," *Transportation Science*, vol. 42, no. 2, pp. 175–196, 2008.

[2] R. K. Ahuja, K. C. Jha, and J. Liu, "Solving real-life railroad blocking problems," *Interfaces*, vol. 37, no. 5, pp. 404–419, 2007.

[3] C. Barnhart, P. Belobaba, and A. R. Odoni, "Applications of operations research in the air transport industry," *Transportation science*, vol. 37, no. 4, pp. 368–391, 2003.

[4] C. Barnhart, A. Farahat, and M. Lohatepanont, "Airline fleet assignment with enhanced revenue modeling," *Operations research*, vol. 57, no. 1, pp. 231–244, 2009.

[5] L. Bertazzi, M. Savelsbergh, and M. G. Speranza, "Inventory routing," in *The vehicle routing problem: latest advances and new challenges*, Springer, 2008, pp. 49–72.

[6] A. M. Campbell, L. W. Clarke, and M. W. Savelsbergh, "Inventory routing in practice," in *The vehicle routing problem*, SIAM, 2002, pp. 309–330.

[7] G. K. Rand, D. Applegate, R. Bixby, V Chvatal, W. Cook, G Dantzig, R Fulkerson, S Johnson, T Elperin, I. Gertsbakh, *et al.*, "Service network design in freight transportation," *Interfaces*, vol. 43, no. 4, pp. 388–395, 2013.

[8] A. I. Jarrah, E. Johnson, and L. C. Neubert, "Large-scale, less-than-truckload service network design," *Operations Research*, vol. 57, no. 3, pp. 609–625, 2009.

[9] D. Kim, C. Barnhart, K. Ware, and G. Reinhardt, "Multimodal express package delivery: A service network design application," *Transportation Science*, vol. 33, no. 4, pp. 391–407, 1999.

[10] P. Toth and D. Vigo, *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[11] P. Singh and P. Saxena, "The multiple objective time transportation problem with additional restrictions," *European Journal of Operational Research*, vol. 146, no. 3, pp. 460–476, 2003.

[12] D Anuradha and P Pandian, "Solving bottleneck bi-criteria transportation problems," in *International Conference on Mathematical Modelling and Scientific Computation*, Springer, 2012, pp. 114–123.

[13] G. Maity, S. K. Roy, and J. L. Verdegay, "Time variant multi-objective interval-valued transportation problem in sustainable development," *Sustainability*, vol. 11, no. 21, p. 6161, 2019.

[14] S Arora and M. Puri, "On lexicographic optimal solutions in transportation problems," *Optimization*, vol. 39, no. 4, pp. 383–403, 1997.

[15] R. Abounacer, M. Rekik, and J. Renaud, "An exact solution approach for multi-objective location–transportation problem for disaster response," *Computers & Operations Research*, vol. 41, pp. 83–93, 2014.

[16] Q. Bai, A. Ahmed, Z. Li, and S. Labi, "A hybrid pareto frontier generation method for trade-off analysis in transportation asset management," *Computer-Aided Civil and Infrastructure Engineering*, vol. 30, no. 3, pp. 163–180, 2015.

[17] M. Ehrgott and X. Gandibleux, "A survey and annotated bibliography of multiobjective combinatorial optimization," *Or Spectrum*, vol. 22, no. 4, pp. 425–460, 2000.

[18] X. Gandibleux, *Multiple criteria optimization: state of the art annotated bibliographic surveys*. Springer Science & Business Media, 2006, vol. 52.

[19] T. Stidsen, K. A. Andersen, and B. Dammann, "A Branch and Bound Algorithm for a Class of Biobjective Mixed Integer Programs," *Management Science*, vol. 60, no. April, pp. 1009–1032, 2014.

[20] B. Soylu and G. B. Yıldız, "An exact algorithm for biobjective mixed integer linear programming problems," *Computers & Operations Research*, vol. 72, pp. 204–213, 2016.

[21] T. Perini, N. Boland, D. Pecin, and M. Savelsbergh, "A criterion space method for biobjective mixed integer programming: The boxed line method," *INFORMS Journal on Computing*, 2019.

[22] E. Turban, D. King, J. K. Lee, T.-P. Liang, and D. C. Turban, *Electronic commerce: A managerial and social networks perspective*. Springer, 2015.

[23] M. Savelsbergh and T. Van Woensel, "50th anniversary invited article —- City logistics: Challenges and opportunities," *Transportation Science*, vol. 50, no. 2, pp. 579–590, 2016.

[24] T. G. Crainic, "Service network design in freight transportation," *European Journal of Operational Research*, vol. 122, no. 2, pp. 272–288, 2000.

[25] N. Wieberneit, "Service network design for freight transportation: A review," *OR spectrum*, vol. 30, no. 1, pp. 77–112, 2008.

[26] I. Herszterg, Y. Ridouane, N. Boland, A. Erera, and M. Savelsbergh, "Near real-time loadplan adjustments for less-than-truckload carriers," 2020.

[27] P. Belotti, B. Soylu, and M. M. Wiecek, "A branch-and-bound algorithm for biobjective mixed-integer programs," *Optimization Online*, 2013.

[28] C. A. Coello, "An updated survey of ga-based multiobjective optimization techniques," *ACM Computing Surveys (CSUR)*, vol. 32, no. 2, pp. 109–143, 2000.

[29] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[30] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011.

[31] N. Boland, H. Charkhgard, and M. Savelsbergh, "A criterion space search algorithm for biobjective integer programming: The balanced box method," *INFORMS Journal on Computing*, vol. 27, no. 4, pp. 735–754, 2015.

[32] N. Boland, H. Charkhgard, and M. Savelsbergh, "The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs," *European Journal of Operations Research*, vol. 260, pp. 873–885, 2017.

[33] N. Boland, H. Charkhgard, and M. Savelsbergh, "A criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method," *INFORMS Journal on Computing*, vol. 27, no. 4, pp. 597–618, 2015.

[34] B. Soylu, "The search-and-remove algorithm for biobjective mixed-integer linear programming problems," *European Journal of Operational Research*, vol. 268, no. 1, pp. 281–299, 2018.

[35] G. Mavrotas and D. Diakoulaki, "Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming," *Applied mathematics and computation*, vol. 171, no. 1, pp. 53–71, 2005.

[36] T. Vincent, F. Seipp, S. Ruzika, A. Przybylski, and X. Gandibleux, "Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for biobjective case," *Computers and Operations Research*, vol. 40, pp. 498–509, 1 2013.

[37] T. Stidsen, K. A. Andersen, and B. Dammann, "A branch and bound algorithm for a class of biobjective mixed integer programs," *Management Science*, vol. 60, no. 4, pp. 1009–1032, 2014.

[38] L. A. Wolsey and G. L. Nemhauser, "Integer and combinatorial optimization," in, John Wiley & Sons, 2014, ch. 1, pp. 11–12.

[39] G Mavrotas and D Diakoulaki, "A branch and bound algorithm for mixed zero-one multiple objective linear programming," *European Journal of Operational Research*, vol. 107, pp. 530–541, 1998.

[40] A. Fattahi and M. Turkay, "A one direction search method to find the exact nondominated frontier of biobjective mixed-binary linear programming problems," *European Journal of Operational Research*, vol. 266, no. 2, pp. 415–425, 2018.

[41] H. Wu, I. Herszterg, M. Savelsbergh, and Y. Huang, "Service network design for same-day delivery with hub capacity constraints,"

[42] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1 –11, 2013.

[43] H. N. Psaraftis, M. Wen, and C. A. Kontovas, "Dynamic vehicle routing problems: Three decades and counting," *Networks*, vol. 67, no. 1, pp. 3–31, 2016.

[44] A. Sampaio, M. Savelsbergh, L. Veelenturf, and T. V. Woensel, "Chapter 15 - crowd-based city logistics," in *Sustainable Transportation and Smart Logistics*, J. Faulin, S. E. Grasman, A. A. Juan, and P. Hirsch, Eds., Elsevier, 2019, pp. 381 –400.

[45] A. Alnaggar, F. Gzara, and J. H. Bookbinder, "Crowdsourced delivery: A review of platforms and academic literature," *Omega*, pp. 102 –139, 2019.

[46] G. Berbeglia, J.-F. Cordeau, and G. Laporte, "Dynamic pickup and delivery problems," *European Journal of Operational Research*, vol. 202, no. 1, pp. 8 –15, 2010.

[47] P. Bouros, D. Sacharidis, T. Dalamagas, and T. Sellis, "Dynamic pickup and delivery with transfers," in *Advances in Spatial and Temporal Databases*, D. Pfoser, Y. Tao, K. Mouratidis, M. A. Nascimento, M. Mokbel, S. Shekhar, and Y. Huang, Eds., Springer Berlin Heidelberg, 2011, pp. 112–129.

[48] S. C. Ho, W. Szeto, Y.-H. Kuo, J. M. Leung, M. Petering, and T. W. Tou, "A survey of dial-a-ride problems: Literature review and recent developments," *Transportation Research Part B: Methodological*, vol. 111, pp. 395–421, 2018.

[49] C. Barnhart, N. Krishnan, D. Kim, and K. Ware, "Network design for express shipment delivery," *Computational Optimization and Applications*, vol. 21, no. 3, pp. 239–262, 2002.

[50] B. Yildiz and M. Savelsbergh, "Optimizing package express operations in China," *Optimization Online 6799*, 2019.

[51] B. Lin, Y. Zhao, and R. Lin, "Optimization for courier delivery service network design based on frequency delay," *Computers & Industrial Engineering*, vol. 139, pp. 106 –144, 2020.

[52] M. P. Helme, "Reducing air traffic delay in a space-time network," in *[Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, 1992, pp. 236–242.

[53] S. Yan, C.-C. Lu, J.-H. Hsieh, and H.-C. Lin, "A network flow model for the dynamic and flexible berth allocation problem," *Computers & Industrial Engineering*, vol. 81, pp. 65–77, 2015.

[54] C. Barnhart, H. Jin, and P. H. Vance, "Railroad blocking: A network design application," *Operations Research*, vol. 48, no. 4, pp. 603–614, 2000.

[55] N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh, "The continuous-time service network design problem," *Operations Research*, vol. 65, no. 5, pp. 1303–1321, 2017.

[56] F. Clautiaux, S. Hanafi, R. Macedo, M.-É. Voge, and C. Alves, "Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints," *European Journal of Operational Research*, vol. 258, no. 2, pp. 467–477, 2017.

[57] A.-K. Rothenbächer, M. Drexl, and S. Irnich, "Branch-and-price-and-cut for a service network design and hub location problem," *European Journal of Operational Research*, vol. 255, no. 3, pp. 935–947, 2016.

[58] B. Gendron and M. Larose, "Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design," *EURO Journal on Computational Optimization*, vol. 2, no. 1-2, pp. 55–75, 2014.

[59] B. Gendron, S. Hanafi, and R. Todosijević, "Matheuristics based on iterative linear programming and slope scaling for multicommodity capacitated fixed charge network design," *European Journal of Operational Research*, vol. 268, no. 1, pp. 70–81, 2018.

[60] T. G. Crainic, M. Hewitt, M. Toulouse, and D. M. Vu, "Service network design with resource constraints," *Transportation Science*, vol. 50, no. 4, pp. 1380–1393, 2014.

[61] B. D. Brouer, G. Desaulniers, and D. Pisinger, "A matheuristic for the liner shipping network design problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 72, pp. 42–59, 2014.

[62] K. Lindsey, A. Erera, and M. Savelsbergh, "Improved integer programming-based neighborhood search for less-than-truckload load plan design," *Transportation Science*, vol. 50, no. 4, pp. 1360–1379, 2016.

[63] M. Gendreau, J.-Y. Potvin, A. Smires, and P. Soriano, "Multi-period capacity expansion for a local access telecommunications network," *European Journal of Operational Research*, vol. 172, no. 3, pp. 1051–1066, 2006.

[64] N.-S. Hsu, W.-C. Cheng, W.-M. Cheng, C.-C. Wei, and W. W.-G. Yeh, "Optimization and capacity expansion of a water distribution system," *Advances in Water Resources*, vol. 31, no. 5, pp. 776–786, 2008.

[65] T. V. Mathew and S. Sharma, "Capacity expansion problem for large urban transportation networks," *Journal of Transportation Engineering*, vol. 135, no. 7, pp. 406–415, 2009.

[66] A. Marin and P. Jaramillo, "Urban rapid transit network capacity expansion," *European Journal of Operational Research*, vol. 191, no. 1, pp. 45–60, 2008.

[67] J. Andre, F. Bonnans, and L. Cornibert, "Optimization of capacity expansion planning for gas transportation networks," *European Journal of Operational Research*, vol. 197, no. 3, pp. 1019–1027, 2009.

[68] H. Luss, "A capacity-expansion model for two facility types," *Naval Research Logistics Quarterly*, vol. 26, no. 2, pp. 291–303, 1979.

[69] C. Jablonowski, H. Ramachandran, L. Lasdon, *et al.*, "Modeling facility-expansion options under uncertainty," *SPE Projects, Facilities & Construction*, vol. 6, no. 04, pp. 239–247, 2011.

[70] K.-J. Wang and S.-H. Lin, "Capacity expansion and allocation for a semiconductor testing facility under constrained budget," *Production Planning & Control*, vol. 13, no. 5, pp. 429–437, 2002.

[71] H. Luss, "Operations research and capacity expansion problems: A survey," *Operations research*, vol. 30, no. 5, pp. 907–947, 1982.

[72] S. G. Nurre, B. Cavdaroglu, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace, "Restoring infrastructure systems: An integrated network design and scheduling (inds) problem," *European Journal of Operational Research*, vol. 223, no. 3, pp. 794–806, 2012.

[73] J. Jeong and D. Culler, "Incremental network programming for wireless sensors," in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, IEEE, 2004, pp. 25–33.

[74] R. DeBlasio and C. Tom, "Standards for the smart grid," in *2008 IEEE Energy 2030 Conference*, IEEE, 2008, pp. 1–7.

[75] H. Farhangi, "The path of the smart grid," *IEEE power and energy magazine*, vol. 8, no. 1, pp. 18–28, 2009.

[76] B. Cavdaroglu, E. Hammel, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace, "Integrating restoration and scheduling decisions for disrupted interdependent infrastructure systems," *Annals of Operations Research*, vol. 203, no. 1, pp. 279–294, 2013.

[77] M. Çelik, "Network restoration and recovery in humanitarian operations: Framework, literature review, and research directions," *Surveys in Operations Research and Management Science*, vol. 21, no. 2, pp. 47–61, 2016.

[78] T. Kalinowski, D. Matsypura, and M. W. Savelsbergh, "Incremental network design with maximum flows," *European Journal of Operational Research*, vol. 242, no. 1, pp. 51–62, 2015.

[79] M. Baxter, T. Elgindy, A. T. Ernst, T. Kalinowski, and M. W. Savelsbergh, "Incremental network design with shortest paths," *European Journal of Operational Research*, vol. 238, no. 3, pp. 675–684, 2014.

[80] K. Engel, T. Kalinowski, and M. W. Savelsbergh, "Incremental network design with minimum spanning trees.," *J. Graph Algorithms Appl.*, vol. 21, no. 4, pp. 417–432, 2017.

[81] S. G. Nurre and T. C. Sharkey, "Integrated network design and scheduling problems with parallel identical machines: Complexity results and dispatching rules," *Networks*, vol. 63, no. 4, pp. 306–326, 2014.